Oracle® Database Oracle Text Reference





Oracle Database Oracle Text Reference, 23ai

F46835-09

Copyright © 2005, 2025, Oracle and/or its affiliates.

Primary Author: Binika Kumar

Contributing Authors: Doug Williams, Prakash Jashnani

Contributors: Ajay Sunnyhith Chidurala, Aleksandra Czarlinska, Asha Makur, Bonnie Xia, Ce Wei, Denis Mukhin, Edwin Balthes, Gaurav Yadav, George Krupka, Harichandan Roy, Madhupriya Ravishankar, Mohammad Faisal, Nilay Panchal, Paul Lane, Rahul Kadwe, Rodrigo Fuentes Hernandez, Roger Ford, Sanoop Sethumadhavan, Saurabh Naresh Netravalkar, Sebastian DeLaHoz, Simona Herdan, Sudhir Kumar, Yiming Qi

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software, software documentation, data (as defined in the Federal Acquisition Regulation), or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software," "commercial computer software documentation," or "limited rights data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle®, Java, MySQL, and NetSuite are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Contents

Audience	XX
Documentation Accessibility	XX
Diversity and Inclusion	XX
Related Documents	xx
Conventions	XX
Oracle Text SQL Statements and Operators	
1.1 ALTER INDEX	1-1
1.2 ALTER TABLE: Supported Partitioning Statements	1-33
1.3 CATSEARCH	1-38
1.4 CONTAINS	1-44
1.5 CREATE INDEX	1-55
1.6 CREATE SEARCH INDEX	1-82
1.7 CREATE HYBRID VECTOR INDEX	1-108
1.8 DROP INDEX	1-119
1.9 MATCHES	1-119
1.10 MATCH_SCORE	1-121
1.11 SCORE	1-121
Oracle Text Indexing Elements	
2.1 Overview	2-1
2.2 Creating Preferences	2-2
2.3 Datastore Types	2-2
2.3.1 DIRECT_DATASTORE	2-3
2.3.2 MULTI_COLUMN_DATASTORE	2-4
2.3.2.1 MULTI_COLUMN_DATASTORE Attributes	2-4
2.3.2.2 Indexing and DML	2-5
2.3.2.3 MULTI_COLUMN_DATASTORE Restriction	2-5
2.3.2.4 MULTI COLUMN DATASTORE Example	2-5



2.3.2.6

Tagging Behavior

2-6

2.3.2.7	Indexing Columns as Sections	2-6
2.3.3 DETA	AIL_DATASTORE	2-7
2.3.3.1	DETAIL_DATASTORE Attributes	2-8
2.3.3.2	Synchronizing Primary/Detail Indexes	2-8
2.3.3.3	Example Primary/Detail Tables	2-8
2.3.4 FILE	_DATASTORE	2-10
2.3.4.1	FILE_DATASTORE Attributes	2-10
2.3.4.2	FILE_DATASTORE and Security	2-11
2.3.4.3	FILE_DATASTORE Example	2-12
2.3.5 DIRE	ECTORY_DATASTORE	2-12
2.3.5.1	DIRECTORY_DATASTORE Attributes	2-13
2.3.5.2	DIRECTORY_DATASTORE Example	2-13
2.3.6 URL	_DATASTORE	2-14
2.3.6.1	URL_DATASTORE URL Syntax	2-14
2.3.6.2	URL_DATASTORE Attributes	2-15
2.3.6.3	URL_DATASTORE and Security	2-16
2.3.6.4	URL_DATASTORE Example	2-16
2.3.7 NET	WORK_DATASTORE	2-17
2.3.7.1	NETWORK_DATASTORE URL Syntax	2-17
2.3.7.2	NETWORK_DATASTORE Attributes	2-18
2.3.7.3	NETWORK_DATASTORE Example	2-19
2.3.8 USE	R_DATASTORE	2-20
2.3.8.1	USER_DATASTORE Attributes	2-20
2.3.8.2	USER_DATASTORE Constraints	2-21
2.3.8.3	USER_DATASTORE Editing Procedure after Indexing	2-21
2.3.8.4	USER_DATASTORE with CLOB Example	2-22
2.3.8.5	USER_DATASTORE with BLOB_LOC Example	2-22
2.3.9 NES	TED_DATASTORE	2-23
2.3.9.1	NESTED_DATASTORE Attributes	2-23
2.3.9.2	NESTED_DATASTORE Example	2-24
2.4 Filter Type:	S	2-25
2.4.1 AUT	O_FILTER	2-25
2.4.1.1	AUTO_FILTER Attributes	2-26
2.4.1.2	AUTO_FILTER and Indexing Formatted Documents	2-26
2.4.1.3	AUTO_FILTER and Explicitly Bypassing Plain Text or HTML in Mixed Format Columns	2-27
2.4.1.4	AUTO_FILTER and Character Set Conversion With AUTO_FILTER	2-28
2.4.2 NUL	L_FILTER	2-28
2.4.3 MAIL	_ FILTER	2-29
2.4.3.1	MAIL_FILTER Attributes	2-29
2.4.3.2	MAIL_FILTER Behavior	2-30
2.4.3.3	About the Mail Filter Configuration File	2-30



2.4.3	4 Mail_Filter Example	2-31
2.4.4 U	SER_FILTER	2-32
2.4.4	1 USER_FILTER Attributes	2-32
2.4.4	2 Using USER_FILTER with Charset and Format Columns	2-33
2.4.4	3 USER_FILTER and Explicitly Bypassing Plain Text or HTML in Mixed Format Columns	2-33
2.4.4	4 Character Set Conversion with USER_FILTER	2-34
2.4.4	5 User Filter Example	2-34
2.4.5 P	ROCEDURE_FILTER	2-35
2.4.5	1 PROCEDURE_FILTER Attributes	2-35
2.4.5	2 PROCEDURE_FILTER Parameter Order	2-37
2.4.5	3 PROCEDURE_FILTER Execute Requirements	2-37
2.4.5	4 PROCEDURE_FILTER Error Handling	2-37
2.4.5	5 PROCEDURE_FILTER Preference Example	2-38
2.5 Lexer T	ypes	2-38
2.5.1 A	UTO_LEXER	2-38
2.5.1	1 AUTO_LEXER Language Support	2-39
2.5.1	2 AUTO_LEXER Attributes Inherited from BASIC_LEXER	2-41
2.5.1	3 AUTO_LEXER Language-Independent Attributes	2-41
2.5.1	4 AUTO_LEXER Language-Dependent Attributes	2-43
2.5.1	5 AUTO_LEXER Dictionary Attribute	2-46
2.5.2 B	ASIC_LEXER	2-46
2.5.2	1 BASIC_LEXER Language Support	2-47
2.5.2	2 BASIC_LEXER Attributes	2-48
2.5.2	3 Stemming User-Dictionaries	2-56
2.5.2	4 BASIC_LEXER Example	2-58
2.5.3 N	IULTI_LEXER	2-59
2.5.3	1 MULTI_LEXER Restriction	2-59
2.5.3	2 MULTI_LEXER Multi-language Stoplists	2-60
2.5.3	3 MULTI_LEXER Example	2-60
2.5.3	4 MULTI_LEXER and Querying Multi-Language Tables	2-61
2.5.4 C	HINESE_VGRAM_LEXER	2-61
2.5.5 C	HINESE_LEXER	2-62
2.5.6 J	APANESE_VGRAM_LEXER	2-62
2.5.7 J	APANESE_LEXER	2-64
2.5.8 K	OREAN_MORPH_LEXER	2-65
2.5.8	1 KOREAN_MORPH_ LEXER Dictionaries	2-65
2.5.8	2 KOREAN_MORPH_ LEXER Unicode Support	2-66
2.5.8	3 KOREAN_MORPH_LEXER Attributes	2-67
2.5.8	4 KOREAN_MORPH_ LEXER Limitations	2-67
2.5.8	5 KOREAN_MORPH_LEXER Example: Setting Composite Attribute	2-67
2.5.9 U	SER_LEXER	2-68



2-69 2-69 2-70 2-70 2-72 2-74 2-76 2-80 2-81 2-91
2-70 2-70 2-72 2-73 2-74 2-76 2-80 2-81
2-70 2-72 2-73 2-74 2-76 2-78 2-80 2-81
2-72 2-73 2-74 2-76 2-78 2-80 2-81
2-73 2-74 2-76 2-78 2-80 2-81
2-74 2-76 2-78 2-80 2-81 2-81
2-76 2-78 2-80 2-81 2-81
2-78 2-80 2-81 2-81
2-80 2-81 2-81
2-81 2-81
2-81
2-91
2-91
2-91
2-91
2-92
2-92
2-93
2-100
2-101
2-103
2-103
2-104
2-105
2-105
2-105
2-106
2-106
2-106
2-106
2-106
2-107
2-108
2-109
2-109
2-110
2-110
2-111
2-111
2-112



	2.12.1	Data	a Storage Preferences	2-113
	2.12.2	Filte	r Preferences	2-113
	2.12.3	Lexe	er Preferences	2-113
	2.1	2.3.1	CTXSYS.DEFAULT_LEXER	2-113
	2.1	2.3.2	CTXSYS.DEFAULT_EXTRACT_LEXER	2-114
	2.1	2.3.3	CTXSYS.BASIC_LEXER	2-115
	2.12.4	Sect	tion Group Preferences	2-115
	2.12.5	Stop	olist Preferences	2-115
	2.12.6	Stora	age Preferences	2-115
	2.12.7	Word	dlist Preferences	2-116
	2.13 Syst	tem Pa	arameters	2-116
	2.13.1	Gen	eral System Parameters	2-116
	2.13.2	Defa	ault Index Parameters	2-117
	2.1	3.2.1	CONTEXT Index Parameters	2-117
	2.1	3.2.2	CTXCAT Index Parameters	2-119
	2.1	3.2.3	CTXRULE Index Parameters	2-120
	2.13.3	Defa	ault Policy Parameters	2-120
	2.14 Toke	en Lim	itations for Oracle Text Indexes	2-121
	2.15 Aud	iting O	racle Text DR\$ Index Tables	2-122
	2.15.1	Abou	ut Auditing Oracle Text DR\$ Index Tables	2-122
	2.15.2	Conf	figuring an Oracle Text DR\$ Index Tables Audit Policy	2-123
	2.15.3	Exar	mple: Auditing Update Actions on an Oracle Text DR\$ Index Table	2-123
	2.15.4	How	Oracle Text DR\$ Index Table Entries Appear in the Audit Trail	2-123
3	Oracle 1	ext (CONTAINS Query Operators	
	2.1 Oner	otor Dr	recedence	
	•		recedence	3-2
	3.1.1 3.1.2		o 1 Operators	3-2 3-2
			o 2 Operators and Characters	
	3.1.3		edural Operators	3-3
	3.1.4		edence Examples	3-3
	3.1.5		ng Precedence	3-3
	3.2 ABOU		4- ()	3-4
		JMulat	te (,)	3-6
	3.4 AND	. ,	*** (DT DTC DTD DT)	3-8
			m (BT, BTG, BTP, BTI)	3-9
			RCACHE	3-10
		NEME		3-15
		NESC		3-15
	_	Valend	Ce (=)	3-19
	3.10 Fuz	zy		3-20

2.12 System-Defined Preferences



2-112

	3.11	HASPATH	3-21
	3.12	INPATH	3-23
	3.13	MDATA	3-28
	3.14	MINUS (-)	3-30
	3.15	MNOT	3-30
	3.16	Narrower Term (NT, NTG, NTP, NTI)	3-31
	3.17	NDATA	3-33
	3.18	NEAR (;)	3-35
	3.19	NEAR2	3-39
	3.20	NOT (~)	3-40
	3.21	OR ()	3-41
	3.22	Preferred Term (PT)	3-41
	3.23	Related Term (RT)	3-42
	3.24	SDATA	3-43
	3.25	soundex (!)	3-45
	3.26	stem (\$)	3-46
	3.27	Stored Query Expression (SQE)	3-47
	3.28	SYNonym (SYN)	3-47
	3.29	threshold (>)	3-48
	3.30	Translation Term (TR)	3-49
	3.31	Translation Term Synonym (TRSYN)	3-50
	3.32	Top Term (TT)	3-51
	3.33	weight (*)	3-52
	3.34	wildcards (% _)	3-53
	3.35	WITHIN	3-54
	3.36	Supported Oracle Text CONTAINS Query Operators for In-Memory Full Text Search	3-59
4	Spe	cial Characters in Oracle Text Queries	
	4.1	Grouping Characters	4-1
	4.2	Escape Characters	4-1
	4.3	Reserved Words and Characters	4-2
5	CT	K_ADM Package	
	5.1	About CTX_ADM Package Procedures	5-1
	5.2	MARK_FAILED	5-1
	5.3	RECOVER	5-3
	5.4	RESET_AUTO_OPTIMIZE_STATUS	5-3
	5.5	SET_PARAMETER	5-3



CT	X_ANL Package	
6.1	About CTX_ANL Package Procedures	6-1
6.2	ADD_DICTIONARY	6-1
6.3	DROP_DICTIONARY	6-4
СТ	X_CLS Package	
7.1	About CTX_CLS Package Procedures	7-1
7.2	TRAIN	7-1
7.3	CLUSTERING	7-5
7.4	SA_TRAIN_MODEL	7-8
7.5	SA_DROP_MODEL	7-9
CT	X_DDL Package	
8.1	ADD_ATTR_SECTION	8-3
8.2	ADD_AUTO_OPTIMIZE	8-4
8.3	ADD_FIELD_SECTION	8-5
8.4	ADD_INDEX	8-8
8.5	ADD_MDATA	8-10
8.6	ADD_MDATA_COLUMN	8-12
8.7	ADD_MDATA_SECTION	8-13
8.8	ADD_NDATA_SECTION	8-14
8.9	ADD_PATH	8-15
8.10	ADD_SDATA_COLUMN	8-17
8.11	ADD_SDATA_SECTION	8-19
8.12	ADD_SEC_GRP_ATTR_VAL	8-23
8.13	ADD_SPECIAL_SECTION	8-24
8.14	ADD_STOPCLASS	8-25
8.15	ADD_STOP_SECTION	8-26
8.16	ADD_STOPTHEME	8-28
8.17	ADD_STOPWORD	8-28
8.18	B ADD_SUB_LEXER	8-30
8.19	ADD_ZONE_SECTION	8-32
8.20	COPY_POLICY	8-35
8.21	. CREATE_INDEX_SET	8-35
8.22	CREATE_PATH_LIST	8-36
8.23	CREATE_POLICY	8-39
8.24	CREATE_PREFERENCE	8-41
8.25	CREATE_SECTION_GROUP	8-43
8.26	CREATE_SHADOW_INDEX	8-46



8.27	CREATE_STOPLIST	8-48
8.28	DROP_INDEX_SET	8-50
8.29	DROP_PATH_LIST	8-50
8.30	DROP_POLICY	8-51
8.31	DROP_PREFERENCE	8-51
8.32	DROP_SECTION_GROUP	8-52
8.33	DROP_SHADOW_INDEX	8-52
8.34	DROP_STOPLIST	8-53
8.35	EXCHANGE_SHADOW_INDEX	8-53
8.36	LOAD_STOPLIST	8-55
8.37	OPTIMIZE_INDEX	8-57
8.38	POPULATE_PENDING	8-63
8.39	PREFERENCE_IMPLICIT_COMMIT	8-63
8.40	RECREATE_INDEX_ONLINE	8-64
8.41	REM_SEC_GRP_ATTR_VAL	8-70
8.42	REMOVE_AUTO_OPTIMIZE	8-71
8.43	REMOVE_INDEX	8-71
8.44	REMOVE_MDATA	8-72
8.45	REMOVE_SECTION	8-73
8.46	REMOVE_STOPCLASS	8-74
8.47	REMOVE_STOPTHEME	8-75
8.48	REMOVE_STOPWORD	8-75
8.49	REMOVE_SUB_LEXER	8-76
8.50	REPLACE_INDEX_METADATA	8-76
8.51	SET_ATTRIBUTE	8-77
8.52	SET_SEC_GRP_ATTR	8-78
8.53	SET_SECTION_ATTRIBUTE	8-79
8.54	SYNC_INDEX	8-81
8.55	UNSET_ATTRIBUTE	8-84
8.56	UNSET_SEC_GRP_ATTR	8-84
8.57	UPDATE_SUB_LEXER	8-85
8.58	UPDATE_POLICY	8-85
8.59	UPDATE_SDATA	8-86
CTX	X_DOC Package	
9.1	About CTX_DOC Package Procedures	9-2
9.2	FILTER	9-2
9.3	GIST	9-5
9.4	HIGHLIGHT	9-8
9.5	IFILTER	9-12
9.6	MARKUP	9-13



9

	9.7	PKENCODE	9-19
	9.8	POLICY_FILTER	9-19
	9.9	POLICY_GIST	9-20
	9.10	POLICY_HIGHLIGHT	9-22
	9.11	POLICY_LANGUAGES	9-24
	9.12	POLICY_MARKUP	9-24
	9.13	POLICY_NOUN_PHRASES	9-27
	9.14	POLICY_PART_OF_SPEECH	9-29
	9.15	POLICY_SNIPPET	9-31
	9.16	POLICY_STEMS	9-33
	9.17	POLICY_THEMES	9-34
	9.18	POLICY_TOKENS	9-36
	9.19	SENTIMENT	9-37
	9.20	SENTIMENT_AGGREGATE	9-38
	9.21	SET_KEY_TYPE	9-40
	9.22	SNIPPET	9-40
	9.23	THEMES	9-44
	9.24	TOKENS	9-47
10	CTX	X_ENTITY Package	
	10.1	ADD_EXTRACT_RULE	10-1
	10.2		10-5
	10.3		10-6
	10.4	CREATE_EXTRACT_POLICY	10-7
	10.5		10-8
	10.6	EXTRACT	10-9
	10.7	IMPORT_DICTIONARY	10-10
	10.8	_	10-12
	10.9		10-13
11	CT)	X_OUTPUT Package	
	11.1	ADD_EVENT	11-1
	11.2	ADD_TRACE	11-2
	11.3	_	11-3
	11.4		11-4
	11.5		11-5
	11.6	_	11-5
	11.7	_ ` _	11-5
	11.8		11-6
	11.9	LOGFILENAME	11-7



	11.10 REMOVE_EVENT	11-7
	11.11 REMOVE_TRACE	11-7
	11.12 RESET_TRACE	11-8
	11.13 START_LOG	11-8
	11.14 START_QUERY_LOG	11-9
12	CTX_QUERY Package	
	12.1 BROWSE_WORDS	12-1
	12.2 COUNT_HITS	12-3
	12.3 EXPLAIN	12-4
	12.4 HFEEDBACK	12-7
	12.5 REMOVE_SQE	12-10
	12.6 RESULT_SET	12-11
	12.7 RESULT_SET_CLOB_QUERY	12-34
	12.8 RESULT_SET_DOCUMENT	12-35
	12.9 STORE_SQE	12-35
	12.10 WARM_CACHE	12-37
13	CTX_REPORT Package	
	13.1 Description of Procedures in CTX_REPORT	13-1
	13.2 Using the Function Versions	13-2
	13.3 DESCRIBE_INDEX	13-2
	13.4 DESCRIBE_POLICY	13-3
	13.5 CREATE_INDEX_SCRIPT	13-4
	13.6 CREATE_POLICY_SCRIPT	13-4
	13.7 INDEX_SIZE	13-5
	13.8 INDEX_STATS	13-6
	13.9 QUERY_LOG_SUMMARY	13-13
	13.10 SHOW_TOKENS	13-17
	13.11 TOKEN_INFO	13-20
	13.12 TOKEN_TYPE	13-22
	13.13 VALIDATE_INDEX	13-23
14	CTX_THES Package	
	14.1 ALTER_PHRASE	14-2
	14.2 ALTER_THESAURUS	14-3
	14.3 BT	14-4
	14.4 BTG	14-6
	14.5 BTI	14-7



	14.6	ВТР	14-8
	14.7	CREATE_PHRASE	14-9
	14.8	CREATE_RELATION	14-10
	14.9	CREATE_THESAURUS	14-11
	14.10	CREATE_TRANSLATION	14-12
	14.11	DROP_PHRASE	14-13
	14.12	DROP_RELATION	14-13
	14.13	DROP_THESAURUS	14-15
	14.14	DROP_TRANSLATION	14-15
	14.15	EXPORT_THESAURUS	14-16
	14.16	HAS_RELATION	14-17
	14.17	IMPORT_THESAURUS	14-17
	14.18	NT	14-18
	14.19	NTG	14-20
	14.20	NTI	14-21
	14.21	NTP	14-23
	14.22	OUTPUT_STYLE	14-24
	14.23	PT	14-24
	14.24	RT	14-26
	14.25	SN	14-27
	14.26	SYN	14-27
	14.27	THES_TT	14-29
	14.28	TR	14-30
	14.29	TRSYN	14-32
	14.30	TT	14-33
	14.31	UPDATE_TRANSLATION	14-34
15	СТХ	_ULEXER Package	
	15.1	WILDCARD_TAB	15-1
1.6	DDM	AS SEADCH Dackago	
16		IS_SEARCH Package	
	16.1	CREATE_INDEX	16-1
	16.2	ADD_SOURCE	16-3
	16.3	REMOVE_SOURCE	16-4
	16.4	DROP_INDEX	16-5
	16.5	GET_DOCUMENT	16-5
	16.6	FIND	16-6



17 Oracle Text Utilities

17.1 The	saurus Loader (ctxload)	17-1
17.1.1	ctxload Text Loading	17-1
17.1.2	ctxload Syntax	17-2
17.1.3	ctxload Examples	17-3
17.2 Enti	ity Extraction User Dictionary Loader (ctxload)	17-4
17.2.1	ctxload Syntax	17-4
17.2.2	Considerations When Creating a User Dictionary	17-4
17.2.3	XML Schema	17-5
17.2.4	ctxload Example	17-6
17.3 Kno	wledge Base Extension Compiler (ctxkbtc)	17-6
17.3.1	Knowledge Base Character Set	17-7
17.3.2	ctxkbtc Syntax	17-7
17.3.3	ctxkbtc Usage Notes	17-8
17.3.4	ctxkbtc Limitations	17-8
17.3.5	ctxkbtc Constraints on Thesaurus Terms	17-9
17.3.6	ctxkbtc Constraints on Thesaurus Relations	17-9
17.3.7	Extending the Knowledge Base	17-9
17.3.8	Example for Extending the Knowledge Base	17-10
17.3.9	Adding a Language-Specific Knowledge Base	17-10
17.3.10	Limitations for Adding a Knowledge Base	17-11
17.3.11	Order of Precedence for Multiple Thesauri	17-11
17.3.12	2 Size Limits for Extended Knowledge Base	17-11
17.4 Lex	ical Compiler (ctxlc)	17-11
17.4.1	Syntax of ctxlc	17-12
17.4.2	ctxlc Performance Considerations	17-13
17.4.3	ctxlc Usage Notes	17-13
17.4.4	ctxlc Example	17-13
Oracle 7	Гехt Alternative Spelling	
18.1 Ove	erview of Alternative Spelling Features	18-1
18.1.1	Alternate Spelling	18-2
18.1.2	Base-Letter Conversion	18-2
18.1.3	New German Spelling	18-2
18.2 Ove	erriding Alternative Spelling Features	18-3
18.3 Alte	rnative Spelling Conventions	18-3
18.3.1	German Alternate Spelling Conventions	18-4
18.3.2	Danish Alternate Spelling Conventions	18-4
18.3.3	Swedish Alternate Spelling Conventions	18-4



18

A Oracle Text Result Tables

A.1 CTX	(_QUERY Result Tables	A-1
A.1.1	EXPLAIN Table	A-1
A.	1.1.1 EXPLAIN Table Structure	A-1
A.	1.1.2 EXPLAIN Table Operation Column Values	A-2
A.	1.1.3 EXPLAIN Table OPTIONS Column Values	A-3
A.1.2	HFEEDBACK Table	A-3
A.	1.2.1 HFEEDBACK Table Structure	A-3
A.	1.2.2 HFEEDBACK Table Operation Column Values	A-4
A.	1.2.3 HFEEDBACK Table OPTIONS Column Values	A-4
A.	1.2.4 CTX_FEEDBACK_TYPE	A-5
A.2 CTX	C_DOC Result Tables	A-6
A.2.1	Filter Table	A-6
A.2.2	Gist Table	A-6
A.2.3	Highlight Table	A-7
A.2.4	Markup Table	A-7
A.2.5	Theme Table	A-7
A.2.6	Token Table	A-8
A.3 CTX	C_THES Result Tables and Data Types	A-8
	EXP_TAB Table Type	A-8
3.1 Abo	ut Document Filtering Technology	B-1
B.1.1	Latest Updates for Patch Releases	B-1
B.1.2	Restrictions on Format Support	
B.1.3	restrictions on remar support	
B.1.4	Supported Platforms for AUTO_FILTER Technology	B-2
B.1.5	• •	B-2 B-2
2.1.0	Supported Platforms for AUTO_FILTER Technology	B-2 B-2 B-3
B.1.6	Supported Platforms for AUTO_FILTER Technology Filtering on PDF Documents and Security Settings	B-2 B-2 B-3 B-4
	Supported Platforms for AUTO_FILTER Technology Filtering on PDF Documents and Security Settings PDF Filtering Limitations	B-2 B-2 B-3 B-4 B-4
B.1.6 B.1.7	Supported Platforms for AUTO_FILTER Technology Filtering on PDF Documents and Security Settings PDF Filtering Limitations Environment Variables	B-2 B-2 B-3 B-4 B-4
B.1.6 B.1.7	Supported Platforms for AUTO_FILTER Technology Filtering on PDF Documents and Security Settings PDF Filtering Limitations Environment Variables General Limitations	B-2 B-3 B-4 B-4 B-4 B-4
B.1.6 B.1.7 B.2 Sup	Supported Platforms for AUTO_FILTER Technology Filtering on PDF Documents and Security Settings PDF Filtering Limitations Environment Variables General Limitations ported Document Formats	B-2 B-2 B-3 B-4 B-4 B-4
B.1.6 B.1.7 B.2 Sup B.2.1	Supported Platforms for AUTO_FILTER Technology Filtering on PDF Documents and Security Settings PDF Filtering Limitations Environment Variables General Limitations ported Document Formats Archive File Format	B-2 B-3 B-4 B-4 B-4 B-5 B-5
B.1.6 B.1.7 B.2 Sup B.2.1 B.2.2	Supported Platforms for AUTO_FILTER Technology Filtering on PDF Documents and Security Settings PDF Filtering Limitations Environment Variables General Limitations ported Document Formats Archive File Format Database Formats	B-2 B-3 B-4 B-4 B-4 B-5 B-5
B.1.6 B.1.7 B.2 Sup B.2.1 B.2.2 B.2.3	Supported Platforms for AUTO_FILTER Technology Filtering on PDF Documents and Security Settings PDF Filtering Limitations Environment Variables General Limitations ported Document Formats Archive File Format Database Formats E-Book Formats	B-2 B-3 B-4 B-4 B-4 B-4 B-5
B.1.6 B.1.7 B.2 Sup B.2.1 B.2.2 B.2.3 B.2.4	Supported Platforms for AUTO_FILTER Technology Filtering on PDF Documents and Security Settings PDF Filtering Limitations Environment Variables General Limitations ported Document Formats Archive File Format Database Formats E-Book Formats Email Formats	B-2 B-3 B-4 B-4 B-4 B-5 B-5 B-5
B.1.6 B.1.7 B.2 Sup B.2.1 B.2.2 B.2.3 B.2.4 B.2.5	Supported Platforms for AUTO_FILTER Technology Filtering on PDF Documents and Security Settings PDF Filtering Limitations Environment Variables General Limitations ported Document Formats Archive File Format Database Formats E-Book Formats E-Book Formats Graphic Formats (Raster and Vector Image)	B-2 B-3 B-4 B-4 B-4 B-5 B-5 B-6 B-6
B.1.6 B.1.7 B.2 Sup B.2.1 B.2.2 B.2.3 B.2.4 B.2.5 B.2.6	Supported Platforms for AUTO_FILTER Technology Filtering on PDF Documents and Security Settings PDF Filtering Limitations Environment Variables General Limitations ported Document Formats Archive File Format Database Formats E-Book Formats Email Formats Graphic Formats (Raster and Vector Image) Multimedia Formats	B-2 B-3 B-4 B-4 B-4 B-5 B-5 B-6 B-6 B-8
B.1.6 B.1.7 B.2 Sup B.2.1 B.2.2 B.2.3 B.2.4 B.2.5 B.2.6 B.2.7	Supported Platforms for AUTO_FILTER Technology Filtering on PDF Documents and Security Settings PDF Filtering Limitations Environment Variables General Limitations ported Document Formats Archive File Format Database Formats E-Book Formats Email Formats Graphic Formats (Raster and Vector Image) Multimedia Formats Other Formats	B-2 B-3 B-4 B-4 B-4 B-5 B-5 B-6 B-6 B-8 B-10 B-11



B.2.10	Text and Markup Formats	B-13
B.2.11	Word Processing and Desktop Publishing Formats	B-15
Text Load	ding Examples for Oracle Text	
C.1 SQLI	NSERT Example	C-1
C.2 SQL*I	_oader Example	C-1
C.2.1	Creating the Table	C-1
C.2.2	Issuing the SQL*Loader Command	C-2
C.2.	2.1 Example Control File: loader1.dat	C-2
C.2.	2.2 Example Data File: loader2.dat	C-2
C.3 Struct	ure of ctxload Thesaurus Import File	C-3
C.3.1	Import File Format	C-3
C.3.2	Alternate Hierarchy Structure	C-5
C.3.3	Usage Notes for Terms in Import Files	C-6
C.3.4	Usage Notes for Relationships in Import Files	C-7
C.3.5	Examples of Import Files	C-7
C.3.	5.1 Example 1 (Flat Structure)	C-7
C.3.	5.2 Example 2 (Hierarchical)	C-8
C.3.	5.3 Example 3	C-8
D.1 Introd	uction	D-1
D.2 Indexi	ng	D-1
D.2.1	Multilingual Features for Text Index Types	D-1
D.2.	1.1 CONTEXT Index Type	D-2
D.2.	1.2 CTXCAT Index Type	D-2
D.2.	1.3 CTXRULE Index Type	D-3
D.2.2	Lexer Types	D-3
D.2.3	Basic Lexer Features	D-4
D.2.	3.1 Theme Indexing	D-4
D.2.	3.2 Alternate Spelling	D-4
D.2.	3.3 Base Letter Conversion	D-5
D.2.	3.4 Composite	D-5
D.2.	3.5 Index Stems	D-5
D.2.4	Multi Lexer Features	D-6
D.2.5	World Lexer Features	D-6
D.3 Query	ring	D-8
D.4 Suppl	ied Stoplists	D-8
D.5 Know	ledge Base	D-9
D.5 KIIOW	leage base	Ъ 3



E.1	Scoring Algorithm for Word Queries	E
E.2	Word Scoring Example	E
E.3	DML and Scoring Algorithm	E
Ora	acle Text Views	
F.1	CTX_ALEXER_DICTS	F
F.2	CTX_AUTO_OPTIMIZE_INDEXES	F
F.3	CTX_AUTO_OPTIMIZE_STATUS	F
F.4	CTX_AUTOSYNC_JOBS	F-
F.5	CTX_AUTOSYNC_STATUS	F
F.6	CTX_BACKGROUND_EVENTS	F-
F.7	CTX_CLASSES	F-
F.8	CTX_FILTER_BY_COLUMNS	F-
F.9	CTX_FILTER_CACHE_STATISTICS	F-
F.10	CTX_INDEXES	F-
F.11	CTX_INDEX_ERRORS	F-
F.12	CTX_INDEX_OBJECTS	F-1
F.13	CTX_INDEX_PARTITIONS	F-1
F.14	CTX_INDEX_SETS	F-1
F.15	CTX_INDEX_SET_INDEXES	F-1
F.16	CTX_INDEX_SUB_LEXERS	F-1
F.17	CTX_INDEX_SUB_LEXER_VALUES	F-1
F.18	CTX_INDEX_VALUES	F-1
F.19	CTX_OBJECTS	F-1
F.20	CTX_OBJECT_ATTRIBUTES	F-1
F.21	. CTX_OBJECT_ATTRIBUTE_LOV	F-1
F.22	CTX_ORDER_BY_COLUMNS	F-1
F.23	CTX_PARAMETERS	F-1
F.24	CTX_PREFERENCES	F-1
F.25	CTX_PREFERENCE_VALUES	F-1
F.26	CTX_SECTIONS	F-1
F.27	CTX_SECTION_GROUPS	F-1
F.28	CTX_SQES	F-1
F.29	CTX_STOPLISTS	F-1
F.30	CTX_STOPWORDS	F-1
F.31	. CTX_SUB_LEXERS	F-1
F.32	CTX_THESAURI	F-1



F.33	CTX_THES_PHRASES	F-1/
F.34	CTX_TRACE_VALUES	F-17
F.35	CTX_USER_ALEXER_DICTS	F-17
F.36	CTX_USER_AUTO_OPTIMIZE_INDEXES	F-18
F.37	CTX_USER_AUTOSYNC_JOBS	F-18
F.38	CTX_USER_AUTOSYNC_STATUS	F-19
F.39	CTX_USER_BACKGROUND_EVENTS	F-19
F.40	CTX_USER_EXTRACT_POLICIES	F-22
F.41	CTX_USER_EXTRACT_POLICY_VALUES	F-22
F.42	CTX_USER_EXTRACT_RULES	F-22
F.43	CTX_USER_EXTRACT_STOP_ENTITIES	F-23
F.44	CTX_USER_EXTRACT_TYPE	F-23
F.45	CTX_USER_ FILTER_BY_COLUMNS	F-23
F.46	CTX_USER_INDEXES	F-24
F.47	CTX_USER_INDEX_ERRORS	F-24
F.48	CTX_USER_INDEX_OBJECTS	F-25
F.49	CTX_USER_INDEX_PARTITIONS	F-25
F.50	CTX_USER_INDEX_SETS	F-26
F.51	CTX_USER_INDEX_SET_INDEXES	F-26
F.52	CTX_USER_INDEX_SUB_LEXERS	F-26
F.53	CTX_USER_INDEX_SUB_LEXER_VALS	F-26
F.54	CTX_USER_INDEX_VALUES	F-27
F.55	CTX_USER_ORDER_BY_COLUMNS	F-27
F.56	CTX_USER_PREFERENCES	F-27
F.57	CTX_USER_PREFERENCE_VALUES	F-28
F.58	CTX_USER_SECTIONS	F-28
F.59	CTX_USER_SECTION_GROUPS	F-28
F.60	CTX_USER_SESSION_SQES	F-28
F.61	CTX_USER_SQES	F-29
F.62	CTX_USER_STOPLISTS	F-29
F.63	CTX_USER_STOPWORDS	F-29
F.64	CTX_USER_SUB_LEXERS	F-29
F.65	CTX_USER_THESAURI	F-30
F.66	CTX_USER_THES_PHRASES	F-30
F.67	CTX_VERSION	F-30
F.68	ALL_DBMS_SEARCH_INDEXES	F-31
F.69	ALL_DBMS_SEARCH_INDEX_SOURCES	F-31
F.70	USER_DBMS_SEARCH_INDEXES	F-31
F.71	USER_DBMS_SEARCH_INDEX_SOURCES	F-31



G Stopword Transformations in Oracle Text

G.1	Understanding Stopword Transformations	G-1
G.2	About Stopwords in Phrase Queries	G-2
G.3	Word Transformations	G-2
G.4	AND Transformations	G-2
G.5	OR Transformations	G-3
G.6	ACCUMulate Transformations	G-3
G.7	MINUS Transformations	G-3
G.8	MNOT Transformations	G-4
G.9	NOT Transformations	G-4
G.10	EQUIValence Transformations	G-4
G.11	NEAR Transformations	G-5
G.12	Weight Transformations	G-5
G.13	Threshold Transformations	G-5
G.14	WITHIN Transformations	G-5



Preface

Oracle Text Reference provides reference information for building applications with Oracle Text.

- Audience
- Documentation Accessibility
- Diversity and Inclusion
- Conventions

Audience

This document is intended for application developers and system administrators who maintain an Oracle Text system in an Oracle environment. To use this document, you need experience with Oracle Database, SQL, SQL*Plus, and PL/SQL.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customer access to and use of Oracle support services will be pursuant to the terms and conditions specified in their Oracle order for the applicable services.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

Related Documents

For more information about Oracle Text, see:

Oracle Text Application Developer's Guide

For more information about Oracle Database, see:

- Oracle Database Concepts
- Oracle Database Administrator's Guide
- Oracle Database Utilities
- Oracle Database Performance Tuning Guide
- Oracle Database SQL Tuning Guide
- Oracle Database SQL Language Reference
- Oracle Database Reference
- Oracle Database Development Guide
- Oracle Database Sample Schemas

For more information about PL/SQL, see:

Oracle Database PL/SQL Language Reference

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
monospace	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.



1

Oracle Text SQL Statements and Operators

These are the SQL statements and Oracle Text operators for creating and managing Oracle Text indexes and performing Oracle Text queries.

- ALTER INDEX
- ALTER TABLE: Supported Partitioning Statements
- CATSEARCH
- CONTAINS
- CREATE INDEX
- CREATE HYBRID VECTOR INDEX
- CREATE SEARCH INDEX
- DROP INDEX
- MATCHES
- MATCH SCORE
- SCORE



Starting with Oracle Database 23ai, you can also use the <code>DBMS_SEARCH PL/SQL</code> package to create, manage, or query search indexes for a textual and range-based ubiquitous search. See <code>DBMS_SEARCH Package</code>.

1.1 ALTER INDEX

Use the ALTER INDEX statement to change or rebuild an existing index, such as Oracle Text index, Oracle Text search index, JSON search index, XML search index, or hybrid vector index.



This section describes the ALTER INDEX statement as it pertains to managing an Oracle Text domain index. For a complete description of the ALTER INDEX statement, see *Oracle Database SQL Language Reference*.

ALTER INDEX Purpose

To make changes to or perform maintenance tasks for a CONTEXT, CTXCAT, or CTXRULE index.

Note:

- When you use ALTER INDEX to shift from FAST_DML to FAST_QUERY, you might encounter the DRG-11380 "operation is not allowed on unsynced index" error. To overcome this error, run the SYNC command on the index and then retry ALTER INDEX.
- When you run any DML or query workload during ALTER INDEX, you might encounter an ORA-00060 or other error that may mark the index UNUSABLE. This is because ALTER INDEX behaves like a DDL operation and is not performed online by default. To overcome this error, set the ONLINE parameter in the ALTER INDEX statement.
- The FAST DML and FAST QUERY options are not supported for online operations.

All Index Types

Use ALTER INDEX to perform the following tasks on all Oracle Text index types:

- Rename the index or index partition. See ALTER INDEX RENAME Syntax.
- Add stopwords to the index. See ALTER INDEX REBUILD Syntax.
- Add or remove a sub_lexer, and remove a stopword or set of stopwords for a given symbol (language or language-independent). See ALTER INDEX Sub_Lexer Syntax.
- Rebuild the index using different preferences. Some restrictions apply for the CTXCAT index type. See ALTER INDEX REBUILD Syntax.

Note:

The Oracle Text indextype CTXCAT is deprecated with Oracle Database 23ai. The indextype itself, and it's operator CTXCAT, can be removed in a future release. Both CTXCAT and the use of CTXCAT grammar as an alternative grammar for CONTEXT queries is deprecated. Instead, Oracle recommends that you use the CONTEXT indextype, which can provide all the same functionality, except that it is not transactional. Near-transactional behavior in CONTEXT can be achieved by using SYNC (ON COMMIT) or, preferably, SYNC (EVERY [time-period]) with a short time period.

CTXCAT was introduced when indexes were typically a few megabytes in size. Modern, large indexes, can be difficult to manage with CTXCAT. The addition of index sets to CTXCAT can be achieved more effectively by the use of FILTER BY and ORDER BY columns, or SDATA, or both, in the CONTEXT indextype. CTXCAT is therefore rarely an appropriate choice. Oracle recommends that you choose the more efficient CONTEXT indextype.

CONTEXT and CTXRULE Index Types

Use ALTER INDEX to perform the following tasks on CONTEXT and CTXRULE index types:

- Resume a failed index operation (creation/optimization).
- Add sections and stop sections to the index.



Replace index metadata.



ALTER INDEX REBUILD Syntax to learn more about performing these tasks

Overview of ALTER INDEX Syntax

The syntax for ALTER INDEX is fairly complex. The major divisions are covered in the following sections:

- ALTER INDEX MODIFY PARTITION Syntax: Use this to modify an index partition's metadata.
- ALTER INDEX PARAMETERS Syntax: Use this to modify the parameters of a nonpartitioned index, or to modify all partitions of a local partitioned index, without rebuilding the index.
- ALTER INDEX RENAME Syntax: Use this to rename an index or index partition.
- ALTER INDEX REBUILD Syntax: Use this to rebuild an index or index partition. With this
 statement, you can also replace index metadata; add stopwords, sections, and stop
 sections to an index; and resume a failed operation.

The parameters for ALTER INDEX REBUILD have their own syntax, which is a subset of the syntax for ALTER INDEX. For example, the ALTER INDEX REBUILD PARAMETERS statement can take either REPLACE or RESUME as an argument, and ALTER INDEX REBUILD PARAMETERS ('REPLACE') can take several arguments. Valid examples of ALTER INDEX REBUILD include the following statements:

```
ALTER INDEX REBUILD PARAMETERS ('REPLACE DATASTORE datastore_pref')
ALTER INDEX REBUILD PARAMETERS ('REPLACE WORDLIST wordlist pref')
```

- ALTER INDEX Syntax for JSON Search Index: Use this to modify the JSON search index preferences, such as DATAGUIDE and SEARCH ON.
- ALTER INDEX Syntax for XML Search Index: Use this to modify the XML search index preferences, such as SEARCH ON.
- ALTER INDEX Syntax for Hybrid Vector Index: Use this to modify or rebuild an existing hybrid vector index.

ALTER INDEX MODIFY PARTITION Syntax

Use the following syntax to modify the metadata of an index partition:

```
ALTER INDEX index name MODIFY PARTITION partition name PARAMETER (paramstring)
```

index name

Specify the name of the index whose partition metadata you want to modify.

partition name

Specify the name of the index partition whose metadata you want to modify.

paramstring

The only valid argument here is 'REPLACE METADATA'. This follows the same syntax as ALTER INDEX REBUILD PARTITION PARAMETERS ('REPLACE METADATA'); see the REPLACE METADATA



subsection of the ALTER INDEX REBUILD Syntax section for more information. (The two statements are equivalent. ALTER INDEX MODIFY PARTITION is offered for ease of use, and is the recommended syntax.)

ALTER INDEX PARAMETERS Syntax

The parameter string now supports READ ONLY MDATA. Use the following syntax to modify the parameters either of nonpartitioned or local partitioned indexes, without rebuilding the index. For partitioned indexes, this statement works at the index level, not at the partition level. This statement changes information for the entire index, including all partitions.

ALTER INDEX index name PARAMETERS (paramstring)

paramstring

ALTER INDEX PARAMETERS accepts the following arguments for paramstring:

'REPLACE METADATA'

Replaces current metadata. See the REPLACE METADATA subsection of the ALTER INDEX REBUILD Syntax section for more information.

'ADD MDATA SECTION secname TAG sectag READ ONLY'

Creates non-updatable MDATA sections so that queries on these MDATA sections do not require extra cursors to be opened on \$I table.

'ADD STOPWORD'

Dynamically adds a stopword to an index. See the ADD STOPWORD subsection of the "ALTER INDEX REBUILD Syntax" section for more information.

'ADD FIELD SECTION'

Dynamically adds a field section to an index. See the ADD FIELD subsection of the "ALTER INDEX REBUILD Syntax" section for more information. You can add an unlimited number of field sections.

'ADD ZONE SECTION'

Dynamically adds a zone section to an index. See the ADD ZONE subsection of the "ALTER INDEX REBUILD Syntax" section for more information.

'ADD ATTR SECTION'

Dynamically adds an attribute section to an index. See the ADD ATTR subsection of the ALTER INDEX REBUILD Syntax section for more information.

'ADD SDATA SECTION'

Dynamically adds an SDATA section to an index. An SDATA section can only be added to BASIC, HTML, XML, and NEWS section groups. It supports both global as well as local indexes. New documents synchronized into the index reflect this new preference. The syntax is:

ALTER INDEX index_name PARAMETERS (ADD SDATA SECTION sdata_section_name TAG sdata section tag DATATYPE sdata section datatype);

The datatype can be VARCHAR2, CHAR, NUMBER, DATE, or RAW.

See Adding an SDATA Section for more information.





Documents that were indexed before adding an SDATA section do not reflect this new preference. Rebuild the index in this case.

Each of the above described parameters has an equivalent ALTER INDEX REBUILD PARAMETERS version, except ADD SDATA SECTION.

For example, ALTER INDEX PARAMETERS ('REPLACE METADATA') is equivalent to ALTER INDEX REBUILD PARAMETERS ('REPLACE METADATA'). However, the ALTER INDEX PARAMETERS versions work on either partitioned or nonpartitioned indexes, whereas the ALTER INDEX REBUILD PARAMETERS versions work only on nonpartitioned indexes.

ALTER INDEX RENAME Syntax

Use the following syntax to rename an index or index partition:

```
ALTER INDEX [schema.]index_name RENAME TO new_index_name;

ALTER INDEX [schema.]index name RENAME PARTITION part name TO new part name;
```

[schema.]index_name

Specify the name of the index to rename.

new_index_name

Specify the new name for schema.index. The new_index_name parameter can be no more than 25 bytes, and 21 bytes for a partitioned index in earlier releases of Oracle Database that have not been upgraded to Oracle Database 12c Release 2 (12.2). If you specify a name longer than 25 bytes (or longer than 21 bytes for a partitioned index), then Oracle Text returns an error and the renamed index is no longer valid.



When <code>new_index_name</code> is more than 25 bytes (21 for local partitioned index) and less than 30 bytes, Oracle Text renames the index, even though the system returns an error. To drop the index and associated tables, you must drop <code>new_index_name</code> with the <code>DROP INDEX</code> statement and then re-create and drop <code>index_name</code>.

The upgraded databases that do not have the <code>compatible</code> parameter set to 12.2 can have the <code>new_index_name</code> parameter no more than 30 bytes, and 30 bytes for a partitioned index. The upgraded databases that have the <code>compatible</code> parameter set to 12.2 or new Oracle Database 12c Release 2 (12.2) installations can have the <code>new_index_name</code> parameter no more than 128 bytes, and 128 bytes for a partitioned index.

part_name

Specify the name of the index partition to rename.

new_part_name

Specify the new name for partition.



ALTER INDEX REBUILD Syntax

Use ALTER INDEX REBUILD to rebuild an index, rebuild an index partition, resume a failed operation, replace index metadata, add stopwords to an index, or add sections and stop sections to an index.

The ALTER INDEX REBUILD syntax has its own subsyntax. That is, its parameters have their own syntax. For example, the ALTER INDEX REBUILD PARAMETERS statement can take either REPLACE or RESUME as an argument, and ALTER INDEX REBUILD PARAMETERS ('REPLACE') has several arguments it can take.



You cannot use the ALTER INDEX REBUILD syntax to add or remove the INMEMORY option associated Text index tables.

Valid examples of ALTER INDEX REBUILD include the following statements:

```
ALTER INDEX REBUILD PARALLEL n

ALTER INDEX REBUILD PARAMETERS (REPLACE DATASTORE datastore_pref)

ALTER INDEX REBUILD PARAMETERS (REPLACE WORDLIST wordlist pref)
```

This is the syntax for ALTER INDEX REBUILD:

```
ALTER INDEX [schema.]index [REBUILD] [PARTITION partname] [ONLINE] [PARAMETERS(paramstring)][PARALLEL N];
```

PARTITION partname

Rebuilds the index partition partname. Only one index partition can be built at a time. When you rebuild a partition you can specify only RESUME or REPLACE in paramstring. These operations work only on the partname you specify.

With the REPLACE operation, you can specify MEMORY, STORAGE, and SYNC for each index partition.

Adding Partitions To add a partition to the base table, use the ALTER TABLE SQL statement. When you add a partition to an indexed table, Oracle Text automatically creates the metadata for the new index partition. The new index partition has the same name as the new table partition. If you must change the index partition name, then use ALTER INDEX RENAME.

Splitting or Merging Partitions Splitting or merging a table partition with ALTER TABLE renders the index partitions invalid. You must rebuild them with ALTER INDEX REBUILD.

ONLINE

Enables you to continue to perform updates, insertions, and deletions on a base table. It does not enable you to query the base table. The ${\tt ONLINE}$ keyword can only be used with the Enterprise Edition of Oracle Database.



You can specify REPLACE or RESUME when rebuilding an index or an index partition ${\tt ONLINE}.$



PARAMETERS (paramstring)

Optionally, specify paramstring. If you do not specify paramstring, then Oracle Text rebuilds the index with existing preference settings.



Oracle Text rebuilds the index using metadata values that have been deep-copied into the index. You can use the <code>CTX_REPORT.CREATE_INDEX_SCRIPT</code> procedure to recreate the user preferences. This procedure generates a script with the preferences that are identical to those used in the original Text index. However, the names of the preferences will be system-generated.

The syntax for paramstring is as follows:

```
paramstring =
'REPLACE
     [DATASTORE datastore pref]
     [FILTER filter pref]
     [LEXER lexer pref]
     [WORDLIST wordlist pref]
     [STORAGE storage pref]
     [STOPLIST stoplist]
     [SECTION GROUP section group]
     [MEMORY memsize
     [[POPULATE | NOPOPULATE]
     [INDEX SET index set]
        [METADATA preference new preference]
     [METADATA FORMAT COLUMN format column name]
     [[METADATA] MAINTENANCE AUTO | MAINTENANCE MANUAL]
     [[METADATA] SYNC (MANUAL | EVERY "interval-string" | ON COMMIT)]
     [[METADATA] TRANSACTIONAL | NONTRANSACTIONAL
     [[METADATA] [ASYNCHRONOUS UPDATE | SYNCHRONOUS UPDATE]]
     [[METADATA] OPTIMIZE (MANUAL | AUTO DAILY | EVERY "interval-string")]
|[DATAGUIDE [ON | OFF | ON CHANGE [ADD VC|Function name]]
| [SEARCH ON (NONE | TEXT | TEXT VALUE[(data types)] | VALUE[(data types)] |
TEXT VALUE STRING) ]
| RESUME [memory memsize]
| ADD STOPWORD word [language language]
| ADD ZONE SECTION section name tag tag
| ADD FIELD SECTION section name tag tag [(VISIBLE | INVISIBLE)]
| ADD ATTR SECTION section name tag tag@attr
| ADD STOP SECTION tag'
```

REPLACE [optional_preference_list]

Rebuilds an index. You can optionally specify your own preferences or system-defined preferences.

You can replace only the preferences that are supported for that index type. For instance, you cannot replace index set for a CONTEXT or CTXRULE index. Similarly, for the CTXCAT index type, you can replace lexer, wordlist, storage index set, and memory preferences.

The POPULATE parameter is the default and need not be specified. If you want to empty the index of its contents, then specify NOPOPULATE. Clear an index of its contents when you must rebuild your index incrementally. The NOPOPULATE choice is available for a specific partition of the index, and not just for the entire index.



Note that ALTER INDEX REBUILD creates a populated index by default, unless you explicitly specify the NOPOPULATE keyword. The outputs of CTX_REPORT.CREATE_INDEX_SCRIPT and CTX_REPORT.DESCRIBE_INDEX include the NOPOPULATE keyword for such indexes. If you are rebuilding a partitioned index using the REPLACE parameter, then you can specify

only STORAGE, MEMORY, and NOPOPULATE.

A new wordlist preference SEPARATE_OFFSETS specifies that the token_info in the index is stored as docids only in one place, and offsets is stored only in another place. Refer to *Oracle Text Application Developer's Guide* for information on improved response time using the SEPARATE OFFSETS option of CONTEXT index.

If this procedure modifies the existing index tables for only the following storage attributes of the BASIC STORAGE type (any one of them), then it will not result in re-indexing of data:

- BIG IO
- I INDEX CLAUSE
- I TABLE CLAUSE
- SEPARATE OFFSETS

Note:

- The BIG_IO attribute of the CONTEXT indextype is deprecated with Oracle Database 23ai, and can be disabled or removed in a future release.
- Oracle recommends that you allow this value to be set to its default value of N.
 BIG_IO was introduced to reduce the cost of seeks when index postings
 exceeded 4KB in length. However, the internal code is relatively inefficient, and
 the attribute cannot be combined with newer index options. Seek cost is much
 less relevant for solid state disks or non-volatile memory devices (NVMe), and
 seek cost is irrelevant when postings are cached. This setting is therefore of little
 benefit for most indexes.

REPLACE METADATA preference new_preference

Replaces the existing preference class settings, including SYNC parameters, of the index with the settings from $new_preference$. Only index preferences and attributes are replaced. The index is not rebuilt.

This statement is useful when you want to replace a preference and its attribute settings after the index is built, without re-indexing all data. re-indexing data can require significant time and computing resources.

This statement is also useful for changing the SYNC parameter type, which can be automatic, manual, or on-commit.

The ALTER INDEX REBUILD PARAMETER ('REPLACE METADATA') statement does not work for a local partitioned index at the global level for the index. You cannot, for example, use this syntax to change a global preference, such as filter or lexer type, without rebuilding the index. Use ALTER INDEX PARAMETERS instead to change the metadata of an index at the global level, including all partitions. See ALTER INDEX PARAMETERS Syntax.



Note:

The ALTER INDEX REPLACE METDATA option is essentially a DDL operation (and not an ONLINE operation), so it may fail if there are any concurrent DML operations requesting locks on the underlying table, including gueries. You must perform ALTER INDEX REPLACE METDATA operations during a quiet time on the system when other user operations are not ongoing on the table or index.

When should I use the METADATA keyword? REPLACE METADATA should be used only when the change in index metadata will not lead to an inconsistent index, which can lead to incorrect query results.

For example, use this statement in the following instances:

- To go from a single-language lexer to a multilexer in anticipation of multilingual data. For an example, see Replacing Index Metadata: Changing Single-Lexer to Multilexer.
- To change the WILDCARD MAXTERMS setting in BASIC_WORDLIST.
- To change the SYNC parameter type, which can be automatic, manual, or on-commit.

These changes are safe and will not lead to an inconsistent index that might adversely affect your query results.

WARNING:

The REPLACE METADATA statement can result in inconsistent index data, which can lead to incorrect guery results. As such, Oracle does not recommend using this statement, unless you carefully consider the effect it will have on the consistency of your index data and subsequent gueries.

There can be many instances when changing metadata can result in inconsistent index data. For example, Oracle recommends against using the METADATA keyword after performing the following procedures:

- Changing the USER DATASTORE procedure to a new PL/SQL stored procedure that has different output.
- Changing the BASIC_WORDLIST attribute PREFIX INDEX from NO to YES because no prefixes have been generated for existing documents. Changing it from YES to NO is safe.
- Adding or changing BASIC LEXER printjoin and skipjoin characters, because new gueries with these characters would be lexed differently from how these characters were lexed at index time.
- Do not use REPLACE METADATA with FORWARD INDEX. Instead use REPLACE STORAGE.

In these unsafe cases, Oracle recommends rebuilding the index.

REPLACE [METADATA] MAINTENANCE AUTO | MAINTENANCE MANUAL

Specifies the maintenance type for synchronization of the CONTEXT and search indexes when there are inserts, updates, or deletes to the base table. The maintenance type specified for an index applies to all index partitions.

You can specify one of the following maintenance types:



Maintenance Type	Description
MAINTENANCE AUTO	This is the default method for synchronizing Oracle Text CONTEXT and search indexes.
	This method sets your index to automatic maintenance, that is, the index is automatically synchronized in the background at optimal intervals.
	You do not need to manually configure a SYNC type or set any synchronization interval. The background mechanism automatically determines the synchronization interval and schedules background SYNC.INDEX operations by tracking the DML queue.
	Note : Shadow indexes do not support automatic maintenance. For a complete list of requirements and restrictions to follow in an automatic maintenance mode, see <i>Oracle Text Application Developer's Guide</i> .
MAINTENANCE MANUAL	This method sets your index to manual maintenance. This is a non-automatic maintenance (synchronization) mode in which you can specify SYNC types, such as MANUAL, EVERY, or ON COMMIT.

For guidelines and examples on switching between the MAINTENANCE AUTO and MAINTENANCE MANUAL methods, see *Oracle Text Application Developer's Guide*.

REPLACE [METADATA] SYNC (MANUAL | EVERY "interval-string" | ON COMMIT)

Specifies the SYNC type for synchronization of the CONTEXT and search indexes when there are inserts, updates, or deletes to the base table.

These SYNC settings are applicable only to the indexes that are set to manual maintenance.



By default, the CONTEXT and search indexes run in an automatic maintenance mode (MAINTENANCE AUTO), which means that your DMLs are automatically synchronized into the index in the background at optimal intervals. Therefore, you do not need to manually configure a SYNC method. However, if required, you can do so if you want to modify the default settings for an index.

You can specify one of the following SYNC methods:

SYNC Type	Description
MANUAL	This is the default synchronization method for CONTEXT
	index. In this method, automatic synchronization is not
	provided. You must manually synchronize the index using
	CTX DDL.SYNC INDEX.
	Use MANUAL to disable ON COMMIT and EVERY
	synchronization.



SYNC Type	Description
EVERY interval-string	The default synchronization interval is set to 30 seconds. Automatically synchronize the index at a regular interval specified by the value of <i>interval-string</i> , which takes the same syntax as that for scheduler jobs. Automatic synchronization using EVERY requires that the index creator have CREATE JOB privileges. Ensure that <i>interval-string</i> is set to a considerable time period so that any previous synchronization jobs will have completed. Otherwise, the synchronization job may stop responding. The <i>interval-string</i> argument must be enclosed in double quotation marks (" "). See Enabling Automatic Index Synchronization at Regular Intervals for an example of automatic synchronization syntax.
ON COMMIT	Synchronize the index immediately after a commit. The commit does not return until the sync is complete. Before Oracle Database Release 18c, the synchronization was performed as a separate transaction. There was a time period, usually small, when the data was committed but index changes were not. Starting with Oracle Database Release 18c, the synchronization is performed as part of the same transaction. The operation uses the memory specified with the <i>memory</i> parameter. Before Oracle Database Release 18c, the sync operation had its own transaction context. If the operation failed, the data transaction still committed. Starting with Oracle Database Release 18c, if there is an irrecoverable index synchronization error, the entire data transaction is rolled back. Recoverable (individual row) synchronization errors are logged in the CTX_USER_INDEX_ERRORS view but the transaction still completes. See Viewing Index Errors under CREATE INDEX. ON COMMIT sync works best when the STAGE_ITAB index option is enabled, otherwise it causes significant fragmentation of the main index, requiring frequent OPTIMIZE calls. ON COMMIT sync is the default synchronization method for SEARCH INDEX and JSON search index.
	SEARCH INDEX and JSON search index. See Enabling Automatic Index Synchronization at Regular Intervals for an example of ON COMMIT syntax. See Oracle Text Application Developer's Guide for more information about the STAGE_ITAB option of the CONTEXT index.

Each partition of a locally partitioned index can have its own type of sync: (ON COMMIT, EVERY, or MANUAL). The type of sync specified in primary parameter strings applies to all index partitions unless a partition specifies its own type.

With automatic (EVERY) synchronization, you can specify memory size and parallel synchronization. The syntax is:

... EVERY interval_string MEMORY mem_size PARALLEL paradegree ...



ON COMMIT synchronizations can only be executed serially and at the same memory size as what was specified at index creation.



This command rebuilds the index. When you want to change the SYNC setting without rebuilding the index, use the REBUILD REPLACE METADATA SYNC (MANUAL | ON COMMIT) operation.

REPLACE [METADATA] TRANSACTIONAL | NONTRANSACTIONAL

This parameter enables you to turn the TRANSACTIONAL property on or off. For more information, see TRANSACTIONAL.

Using this parameter only succeeds if there are no rows in the DML pending queue. Therefore, you may need to sync the index before issuing this command.

To turn on the TRANSACTIONAL index property:

```
Or

ALTER INDEX myidx REBUILD PARAMETERS('replace metadata transactional');

To turn off the TRANSACTIONAL index property:

ALTER INDEX myidx REBUILD PARAMETERS('replace metadata nontransactional');

Or

ALTER INDEX myidx REBUILD PARAMETERS('replace metadata nontransactional');
```

REPLACE [METADATA] [ASYNCHRONOUS_UPDATE | SYNCHRONOUS_UPDATE]

When you update the column in a document on which an Oracle Text index is based, that document is marked as invalid for search operations until index synchronization is performed. Enabling asynchronous update for an index enables a document to be searchable even though its index has not yet been synchronized after the index column was updated. Until the index is synchronized, Oracle Text uses the contents of the old document to answer user queries.



Synchronous update is not supported with the ${\tt TRANSACTIONAL}$ option and for updates that cause row movement.

To enable asynchronous update for a Text index:

```
ALTER INDEX idx PARAMETERS ('REPLACE METADATA asynchronous update');
```

To disable asynchronous update for a Text index:

ALTER INDEX idx PARAMETERS ('REPLACE METADATA synchronous update');



Note:

The ASYNCHRONOUS_UPDATE setting of the CONTEXT indextype is deprecated in Oracle Database 23ai, and can be ignored or removed in a future release.

Oracle can ignore or remove this attribute in a future release. Oracle recommends that you allow this value to be set to its default value, <code>SYNCHRONOUS_UPDATE</code>. To avoid unexpected loss of results during updates, use <code>SYNC</code> (ON COMMIT) or <code>SYNC</code> (EVERY <code>[time-period]</code>) with a short time period.

The ASYNCHRONOUS_UPDATE setting was introduced as a workaround for the fact that updates are implemented as "delete followed by insert," and that deletes are immediate (on commit), while inserts are only performed during an index sync. However, this setting is incompatible with several other index options. Oracle recommends that you discontinue its use.

REPLACE [[METADATA] OPTIMIZE (MANUAL | AUTO_DAILY | EVERY "interval-string")] Specify OPTIMIZE to enable automatic background index optimization. You can specify any one of the following OPTIMIZE methods:

OPTIMIZE Type	Description
MANUAL	Provides no automatic optimization. You must manually optimize the index with CTX_DDL.OPTIMIZE_INDEX.
AUTO_DAILY	 When you specify OPTIMIZE (AUTO_DAILY) in the create index parameter list, a repeatedly running optimize token job and a repeatedly running optimize full job are scheduled for each index and partition: The Optimize token job is scheduled to run weekly from 12 A.M. every Saturday night to optimize \$S* tables. This job runs on tables with non-JSON data type (VARCHAR2, CLOB, or BLOB) to optimize the top 10 most fragmented tokens (determined automatically). The Optimize full job is scheduled to run every midnight from 12 A.M. to 3 A.M. except on Saturday night. Jobs that are not started before 3 A.M. are skipped. These skipped jobs are started before the other jobs that are scheduled to run at 12 A.M. the next day. This job runs on tables with JSON data type or the IS JSON check constraint.
	Existing indexes do not have OPTIMIZE (AUTO_DAILY) by default. You must use ALTER INDEX to enable automatic background index optimization.



OPTIMIZE Type	Description
EVERY "interval-string"	Automatically runs at a regular interval specified by the value <i>interval-string</i> , which takes the same syntax as scheduler jobs.
	 The Optimize token job is scheduled for tables with non-JSON data type.
	This job runs optimize token for the top 10 most fragmented tokens at an interval specified by the user.
	 The Optimize full job is scheduled for tables with JSON data type or the IS JSON check constraint.
	This job runs optimize full weekly at 12 A.M. every Saturday night for \$S* tables.
	Ensure that <i>interval-string</i> is set to a considerable time period so that any previous optimize jobs are complete. The <i>interval-string</i> value must be enclosed in double quotes, and any single quote within <i>interval-string</i> must be
	preceded by the escape character with another single quote.
	If multiple indexes use the OPTIMIZE EVERY "interval-
	string" option, then different jobs are created for each index. These jobs are run concurrently.

With AUTO_DAILY | EVERY "interval-string" setting, you can specify parallel optimization. That syntax is:

```
... [AUTO DAILY | EVERY "interval-string"] PARALLEL paradegree ...
```

RESUME [MEMORY memsize]

Resumes a failed index operation. You can optionally specify the amount of memory to use with memsize.



This Alter index operation applies only to context and ctxrule indexes. It does not apply to ctxcat indexes.

ADD STOPWORD word [language language]

Dynamically adds a stopword word to the index.

Index entries for word that existed before this operation are not deleted. However, subsequent queries on word are treated as though it has always been a stopword.

When your stoplist is a multilanguage stoplist, you must specify language.

The index is *not* rebuilt by this statement.

ADD ZONE SECTION section name tag tag

Dynamically adds the zone section <code>section_name</code> identified by tag to the existing index. The added section <code>section_name</code> applies only to documents indexed after this operation. For the change to take effect, you must manually re-index any existing documents that contain the tag.

The index is *not* rebuilt by this statement.



Note:

This alter index operation applies only to context and ctxrule indexes. It does not apply to ctxcat indexes.

See Also:

Notes

ADD FIELD SECTION section_name tag tag [(VISIBLE | INVISIBLE)]

Dynamically adds the field section $section_name$ identified by tag to the existing index. There is no limit to the number of field sections that can be added.

Optionally specify VISIBLE to make the field sections visible. The default is INVISIBLE.

See Also:

CTX_DDL.ADD_FIELD_SECTION for more information on visible and invisible field sections

The added section <code>section_name</code> applies only to documents indexed after this operation. For the change to affect previously indexed documents, you must explicitly re-index the documents that contain the tag.

This statement does not rebuild the index.

Note:

This ALTER INDEX operation applies only to CONTEXT CTXRULE indexes. It does not apply to CTXCAT indexes.

See Also:

Notes

ADD ATTR SECTION section name tag tag@attr

Dynamically adds an attribute section <code>section_name</code> to the existing index. You must specify the XML tag and attribute in the form <code>tag@attr</code>. You can add attribute sections only to XML section groups.

The added attribute section <code>section_name</code> applies only to documents indexed after this operation. For the change to take effect, you must manually re-index any existing documents that contain the tag.

The index is *not* rebuilt by this statement.



This ALTER INDEX operation applies only to CONTEXT CTXRULE indexes. It does not apply to CTXCAT indexes.

See Also:

Notes

ADD STOP SECTION tag

Dynamically adds the stop section identified by tag to the existing index. As stop sections apply only to automatic sectioning of XML documents, the index must use the AUTO_SECTION_GROUP section group. The tag you specify must be case sensitive and unique within the automatic section group or else ALTER INDEX raises an error.

The added stop section tag applies only to documents indexed after this operation. For the change to affect previously indexed documents, you must explicitly re-index the documents that contain the tag.

The text within a stop section can always be searched.

The number of stop sections you can add is unlimited.

The index is *not* rebuilt by this statement.

See Also:

Notes

Note:

This ALTER INDEX operation applies only to CONTEXT indexes. It does not apply to CTXCAT indexes.

PARALLEL n

Using n, you can optionally specify the parallel degree for parallel indexing. This parameter is supported only when you use SYNC, REPLACE, and RESUME in paramstring. The actual degree of parallelism might be smaller depending on your resources.

Parallel indexing can speed up indexing when you have large amounts of data to index and when your operating system supports multiple CPUs.

ALTER INDEX Syntax for JSON Search Index

```
ALTER INDEX [schema.]index REBUILD

PARAMETERS(

[DATAGUIDE ON [CHANGE (ADD_VC | function_name)] | OFF]

[SEARCH_ON (TEXT | TEXT_VALUE[(data_types)] | VALUE[(data_types)] |

TEXT_VALUE_STRING)]

[REMOVE SEARCH_ON VALUE(VARCHAR2)]

);
```



- The REPLACE keyword is not required with the ALTER INDEX REBUILD PARAMETERS statement for changing the JSON search index preferences.
- You cannot change both the JSON and Oracle Text search index preferences in a single ALTER INDEX statement.
- If you specify the JSON search index preferences (such as DATAGUIDE and SEARCH_ON), other preferences in the PARAMETERS clause are not updated. Similarly, if you specify the Oracle Text search index preferences (such as STORAGE and LEXER), the JSON preferences are not updated.

[schema.]index

Specifies the name of JSON search index that you want to modify.

DATAGUIDE ON | OFF

Modifies data guide support for an existing JSON search index. By default, a JSON search index is created without data guide support. If you enable the JSON data guide support, then you can also define change-trigger procedures.



You use the DATAGUIDE clause only for JSON search indexes.

Specify one of the following options:

- ON: Enables data guide support. If you set the value of DATAGUIDE to ON, then you can also define your own PL/SQL procedure or use the predefined change-trigger procedure ADD VC.
 - ADD VC indicates if virtual columns are created based on the data guide.
 - function name specifies the function to be executed when the data guide changes.
- OFF: Disables both the data guide support and change-trigger procedures. Provides only general search-index functionality.



You cannot turn off the DATAGUIDE clause if the SEARCH_ON clause value is set to NONE.

See Change Triggers For Data Guide-Enabled Search Index in *Oracle Database JSON Developer's Guide*.

SEARCH_ON (TEXT | TEXT_VALUE[(data_types)] | VALUE[(data_types)] | TEXT_VALUE_STRING)

Modifies search preferences specified for an existing JSON search index.



You can use the SEARCH ON clause only for JSON and XML search indexes.

You can specify one of the following ${\tt SEARCH_ON}$ options:

Option	Description	
TEXT	Enables full-text search component, which indicates that only textual data is indexed for full-text search queries. This also includes queries that rely on path information. The index is used for JSON_TEXTCONTAINS predicates and for JSON_VALUE or JSON_EXISTS predicates that manipulate strings when using JSON search index. If your queries involve only full-text search and not stringrange search or numeric search, then you can save some index maintenance time and disk space by specifying this option. Example:	
	ALTER INDEX [schema.]index REBUILD PARAMETERS ('SEARCH_ON TEXT);	



Option

Description

VALUE[(data types)]

Enables range-search component for the specified data types. This allows the index to be picked up for predicates using relational operators (>, <, ==, >=, <=, !=). A JSON search index that is created with only SEARCH_ON VALUE cannot answer full-text queries by using the JSON TEXTCONTAINS operator.

Supported data types:

- NUMBER for indexing numeric values.
- TIMESTAMP for indexing date-time values.
- VARCHAR2 for indexing complete string values. The string values are indexed as is without tokenization or other transformations. All the strings that are smaller than or equal to 237 bytes are indexed.

If you do not specify any data type, then the index enables range-search indexing on all supported data types.



The BINARY_DOUBLE data type is allowed only for XML search indexes.

Examples:

• This example specifies the default behavior:

```
ALTER INDEX [schema.]index REBUILD PARAMETERS ('SEARCH_ON VALUE');
```

• These examples explicitly specify data types using the VALUE (data types) syntax:

```
ALTER INDEX [schema.]index REBUILD PARAMETERS ('SEARCH_ON VALUE(NUMBER)');
```

ALTER INDEX [schema.]index REBUILD
PARAMETERS ('SEARCH_ON VALUE (NUMBER,
TIMESTAMP, VARCHAR2)');



Option	Description		
TEXT_VALUE[(data_ty pes)]	Enables both the full-text and range-search components for the specified data types. Supported data types: NUMBER for indexing numeric values.		
	TIMESTAMP for indexing date-time values.		
	 VARCHAR2 for indexing complete string values. The string values are indexed as is without tokenization or other transformations. All the strings that are smaller than or equal to 237 bytes are indexed. 		
	If you do not specify any data type, then the index enables full-text search and range-search indexing on the NUMBER and TIMESTAMP data types.		
	Examples:		
	This example specifies the default behavior:		
	ALTER INDEX [schema.]index REBUILD PARAMETERS('SEARCH_ON TEXT_VALUE');		
	 These examples explicitly specify data types using the TEXT_VALUE(data_types) syntax: 		
	ALTER INDEX [schema.]index REBUILD PARAMETERS('SEARCH_ON TEXT_VALUE(NUMBER)');		
	ALTER INDEX [schema.]index REBUILD PARAMETERS('SEARCH_ON TEXT_VALUE(NUMBER, TIMESTAMP)');		
TEXT_VALUE_STRING	Indicates that text and range-based indexes are created for numeric, date-time, and complete string values. This enables both the full-text and range-search components on the NUMBER, TIMESTAMP, and VARCHAR2 data types. String values are indexed as is without tokenization or other transformations. All the strings that are smaller than or equal to 237 bytes are indexed. Example: ALTER INDEX [schema.]index REBUILD		
	PARAMETERS ('SEARCH_ON TEXT_VALUE_STRING');		

Guidelines for specifying SEARCH_ON transitions:

When you specify the SEARCH_ON clause in the ALTER INDEX REBUILD statement, the system determines both your current configuration and the set of components that you want to enable. The statement then enables any new components and rebuilds the index. If all requested components have already been enabled, this action is the same as an index rebuild. Rebuilding allows the JSON search index to be regenerated with newly enabled indexing and query components.

Note that range-search components of different data types are considered as independent components.



You can disable only the VARCHAR2 range-search component. To disable other components, you must first drop the index using the DROP INDEX statement and then re-create the index with the required components enabled.

Removing indexing components:

You can remove the <code>VARCHAR2</code> data type from any range-search components (<code>VALUE</code>, <code>TEXT_VALUE</code>, or <code>TEXT_VALUE_STRING</code>). Removing the <code>VARCHAR2</code> data type can save you index maintenance time and disk space.

The syntax is:

REMOVE SEARCH ON VALUE (VARCHAR2)

ALTER INDEX Syntax for XML Search Index

```
ALTER INDEX [schema.]index REBUILD

PARAMETERS(

[SEARCH_ON (TEXT | TEXT_VALUE(data_types) | VALUE (data_types))]

[REMOVE SEARCH_ON VALUE(VARCHAR2)]

);
```

[schema.]index

Specifies the name of the XML search index that you want to modify.

SEARCH_ON (TEXT | TEXT_VALUE(data_types) | VALUE(data_types))

Modifies search preferences specified for an existing XML search index.



You can use the SEARCH ON clause only for JSON and XML search indexes.

You can specify one of the following SEARCH ON options:

Option	Description
TEXT	Enables full-text search component, which indicates that only textual data is indexed for full-text search queries. This also includes queries that rely on path information. The index is used for XMLEXISTS predicates that references the XQuery Full Text operators and clauses. If your queries involve only full-text search and not stringrange search or numeric search, then you can save some index maintenance time and disk space by specifying this option. For example:
	ALTER INDEX [schema.]index REBUILD PARAMETERS ('SEARCH_ON TEXT');



Option	Description	
VALUE (data_types)	Enables range-search component for the specified data types. This allows the index to be picked up for predicates using relational operators (>, <, ==, >=, <=, !=). An XML search index that only has the SEARCH_ON VALUE component enabled cannot answer full-text queries, if XQuery Full Text operators are present in an XMLEXISTS predicate. You must specify one or more data types: BINARY_DOUBLE and NUMBER for indexing numeric values. TIMESTAMP for indexing date-time values. VARCHAR2 for indexing complete string values. The string values are indexed as is without tokenization or other transformations. All the strings that are smaller than or equal to 237 bytes are indexed.	
	For example: ALTER INDEX [schema.]index REBUILD PARAMETERS ('SEARCH_ON VALUE(BINARY_DOUBLE)'); ALTER INDEX [schema.]index REBUILD PARAMETERS ('SEARCH_ON	
	VALUE(BINARY_DOUBLE, NUMBER, TIMESTAMP, VARCHAR2)');	
TEXT_VALUE(data_types)	Enables both the full-text and range-search components for the specified data types. For range-search queries, you must specify one or more data types, such as NUMBER (for indexing numeric values) and TIMESTAMP (for indexing date-time values). For example:	
	ALTER INDEX [schema.]index REBUILD PARAMETERS('SEARCH_ON TEXT_VALUE(NUMBER)');	
	ALTER INDEX [schema.]index REBUILD PARAMETERS('SEARCH_ON TEXT_VALUE(NUMBER, TIMESTAMP)');	



You cannot use <code>SEARCH_ON NONE</code> and <code>SEARCH_ON TEXT_VALUE_STRING</code> for an XML search index.

You must explicitly specify a data type with the <code>TEXT_VALUE</code> and <code>VALUE</code> options for an XML search index, otherwise the statement will result in an error.

Guidelines for specifying SEARCH_ON transitions:

When you specify the SEARCH_ON clause in the ALTER INDEX REBUILD statement, the system determines both your current configuration and the set of components that you want to enable. The statement then enables any new components and rebuilds the index. If all requested components have already been enabled, this action is the same as an index rebuild. Rebuilding allows the XML search index to be regenerated with newly enabled indexing and query components.

Note that range-search components of different data types are considered as independent components.

You can disable only the VARCHAR2 range-search component. To disable other components, you must first drop the index using the DROP INDEX statement and then re-create the index with the required components enabled.

Removing indexing components:

You can remove the VARCHAR2 data type from any range-search components (VALUE or TEXT_VALUE). Removing the VARCHAR2 data type can save you index maintenance time and disk space.

The syntax is:

```
REMOVE SEARCH ON VALUE (VARCHAR2)
```

ALTER INDEX Syntax for Hybrid Vector Index

```
ALTER INDEX [schema.]index_name REBUILD

PARAMETERS(

['UPDATE VECTOR INDEX']

['REPLACE vectorizer vectorizer_pref_name']

)

[PARALLEL n];
```

Note:

- If you do not specify the PARAMETERS clause, then all parts of the hybrid vector index (both Oracle Text index and vector index) are recreated with existing preference settings.
- Renaming hybrid vector indexes using the ALTER INDEX RENAME syntax is not supported.
- The ALTER INDEX parameter UDPATE VECTOR INDEX is not supported for Local HVI and HNSW vector indexes.



[schema.]index_name

Specifies name of the hybrid vector index that you want to modify.

PARAMETERS(UPDATE VECTOR INDEX)

Recreates only the vector index part of a hybrid vector index with the original preference settings.

PARAMETERS(REPLACE vectorizer vectorizer_pref_name)

Recreates only the vector index part of a hybrid vector index with the specified vectorizer preference settings.



For non HVI index, replace operation would throw an error, as it cannot replace something that was not present.

PARALLEL

Specifies parallel indexing, as described for the CREATE HYBRID VECTOR INDEX statement. For detailed information on the PARALLEL clause, see CREATE HYBRID VECTOR INDEX.

Examples:

Here are some examples on how you can modify existing hybrid vector indexes:

To rebuild all parts of a hybrid vector index:

Use the following syntax to rebuild all parts of a hybrid vector index (both Oracle Text index and vector index) with the original preference settings:

Syntax:

```
ALTER INDEX index name REBUILD [PARALLEL n];
```

Note that you do not need to specify any PARAMETERS clause when rebuilding both parts of a hybrid vector index.

Example:



To rebuild only the vector index part:

Use the following syntax to rebuild only the vector index part of a hybrid vector index with the original preference settings:

Syntax:

```
ALTER INDEX index name REBUILD
 PARAMETERS ('UPDATE VECTOR INDEX') [PARALLEL n];
Example:
ALTER INDEX my hybrid idx REBUILD
 PARAMETERS ('UPDATE VECTOR INDEX') PARALLEL 3;
SELECT (select id from doc table where rowid = jt.doc rowid) as doc,
      jt.chunk
FROM JSON TABLE (
        DBMS HYBRID VECTOR.SEARCH(
          json(
           '{ "hybrid index_name" : "my_hybrid_idx",
              "vector" :
               },
              "return" :
               { "topN"
                                : 10 }
           }')
          ),
          '$[*]' COLUMNS doc rowid PATH '$.rowid',
                       chunk
                                 PATH '$.chunk text') jt;
```

To recreate indexes with a vectorizer preference:

You can create a vectorizer preference using the <code>DBMS_VECTOR_CHAIN.CREATE_PREFERENCE</code> PL/SQL function. For detailed information on how to create a vectorizer preference, see <code>CREATE_PREFERENCE</code>. After creating the preference, use the <code>REPLACE vectorizer</code> parameter to pass the preference name here.

Syntax:

```
ALTER INDEX index_name REBUILD parameters('REPLACE vectorizer vectorizer_pref_name') [PARALLEL n];
```

Note:

For non-HVI index, the REPLACE operation would throw an error.

Example:

```
ALTER INDEX my_hybrid_idx REBUILD parameters('REPLACE vectorizer my vectorizer pref') [PARALLEL n];
```

To replace only the model and/or vector index type

For an existing HVI index, you can replace the model and/or the index type without specifying the full vectorizer preference using the following syntax.

Syntax:

```
ALTER INDEX schema.index_name REBUILD[
parameters('REPLACE MODEL model_name VECTOR_IDXTYPE hnsw/ivf')];
```

Example:

```
ALTER INDEX schema.my_hybrid_idx REBUILD[
parameters('REPLACE MODEL my_model_name VECTOR_IDXTYPE ivf')];
```

Note:

For non HVI indexes, this syntax would throw an error. If a vectorizer is also specified alongside the model and/or vector_idxtype, it would lead to an error, as only one of either vectorizer or model/vector_idxtype is allowed.

Converting an existing Text SEARCH or JSON SEARCH index to an HVI

You can convert (up-convert) an existing Text SEARCH or JSON SEARCH index to an HVI without a full rebuild of the textual/JSON parts of the index using the following syntax. This syntax is supported only for non-HVI index. After this operation, the index would be modified to become an HVI index. This would mean that all HVI operation would be supported and ctx_report would reflect this update correctly.

Syntax:

```
ALTER INDEX schema.index_name REBUILD [parameters('add vectorizer_fast vectorizer_pref_name')];
```

Example:

```
ALTER INDEX schema.my_hybrid_idx REBUILD [parameters('add vectorizer_fast my_vectorizer_pref')];
```

For other cases that are inapplicable to "up-convert", you can convert an existing Text SEARCH or JSON SEARCH index to an HVI with a full rebuild using the following syntax:

```
ALTER INDEX schema.index_name REBUILD [parameters('add vectorizer vectorizer pref name')];
```



Example:

```
ALTER INDEX schema.my_hybrid_idx REBUILD [parameters('add vectorizer my vectorizer pref)'];
```

For detailed information on managing hybrid vector indexes, see *Oracle Database AI Vector Search User's Guide*.

ALTER INDEX Sub Lexer Syntax

```
New paramstring =
'REPLACE
     [DATASTORE datastore pref]
     [FILTER filter pref]
     [LEXER lexer pref]
     [WORDLIST wordlist pref]
     [STORAGE storage pref]
     [STOPLIST stoplist]
     [SECTION GROUP section_group]
     [MEMORY memsize
     [[POPULATE | NOPOPULATE]
     [INDEX SET index set]
     [METADATA preference new preference]
     [[METADATA] MAINTENANCE AUTO | MAINTENANCE MANUAL]
     [[METADATA] SYNC (MANUAL | EVERY "interval-string" | ON COMMIT)]
     [[METADATA] TRANSACTIONAL|NONTRANSACTIONAL
| RESUME [memory memsize]
| OPTIMIZE [token index token | fast | full [maxtime (time | unlimited)]
| SYNC [memory memsize]
| ADD STOPWORD word [language language][LANGUAGE DEPENDENT(TRUE|FALSE)]
| ADD ZONE SECTION section name tag tag
| ADD FIELD SECTION section name tag tag [(VISIBLE | INVISIBLE)]
| ADD ATTR SECTION section_name tag tag@attr
| ADD STOP SECTION tag
| ADD SUB LEXER sub lexer name LANGUAGE language [ALT VALUE
alternate value for language] [LANGUAGE DEPENDENT (TRUE | FALSE)]
| REMOVE SUB LEXER LANGUAGE language
| REMOVE STOPWORD word [LANGUAGE language]
| REMOVE STOPWORDS FOR LANGUAGE language
| MIGRATE to MULTI STOPLIST [LANGUAGE COLUMN lang]
| MIGRATE FIELD SECTION field_section_name to [READ ONLY] MDATA
| UPDATE SUB LEXER LANGUAGE language TO sub lexer preference
| ADD MDATA SECTION secname TAG sectag READ ONLY
Sub_Lexer Example
ALTER INDEX myidx PARAMETERS('ADD SUB LEXER mycompany lexer LANGUAGE mycompany
LANGUAGE DEPENDENT FALSE');
ALTER INDEX myidx PARAMETERS('REMOVE STOPWORDS FOR LANGUAGE mycompany');
```

Sub Lexer Notes

The language can be Oracle predefined language symbols (globalization support name or abbreviation of an Oracle Text-supported language), or user-defined symbols for language independent sub-lexer or stopword.

ADD SUB LEXER

The following conditions apply:

- If LANGUAGE DEPENDENT clause is not provided, it will default TRUE.
- Sync will be blocked (or it will be blocked by sync).
- If adding first language independent sub_lexer, then base table will also be locked.
- Adding first language independent sub_lexer or stopword will take longer to complete.
 Otherwise, it should take fraction of a second to complete unless it's being blocked by ongoing sync process on the same index.

REMOVE SUB LEXER

Will succeed only if there are no documents with language column set to the symbol for the sub_lexer being removed.

REMOVE STOPWORD

The following conditions apply:

- If LANGUAGE clause is not specified, it is assumed that the index is using basic_stoplist. If the index is not using basic stoplist, an error will be raised.
- If the index is using <code>basic_stoplist</code> (instead of <code>multi_stoplist</code>), then it will succeed only if the base table is empty.
- If the index is using multi_stoplist, and user specifies "ALL" for LANGUAGE clause, then it will succeed only if the base table is empty.
- If the index is using multi_stoplist, and user specifies a symbol for LANGUAGE clause, then it will succeed only if there are no documents with language column set to the symbol for the stopword being removed.

See Also:

ALTER INDEX REBUILD Syntax

MIGRATE TO MULTI_STOPLIST [LANGUAGE COLUMN lang]

The following conditions apply:

- Migrate the stoplist of an existing Text index to multi_stoplist. The language of the
 existing stopwords will have the value of ALL.
- If Language column has already been defined for the index:
 - LANGUAGE COLUMN can be skipped (old language column is retained for the index).
 - If LANGUAGE COLUMN is specified and there is a mismatch between index language column and the one specified, an error will be raised.
- LANGUAGE COLUMN must be specified for the index; otherwise, an error is raised.

MIGRATE FIELD SECTION TO MDATA SECTION

The following conditions apply:

- Allow user to convert a field section to MDATA section. Specify READ ONLY if the MDATA section is meant to be a READ_ONLY MDATA section (ADD and REMOVE not allowed).
- Limitation: Tokens in migrated MDATA sections will not have typical MDATA characteristics - case information, tokens being stored as it is in the document, etc. To retain these, those documents need to be reindexed.



UPDATE SUB_LEXER LANGUAGE SUB_LEXER_SYMBOL TO SUB LEXER PREFERENCE

The following conditions apply:

- Allows user to update sublexer dynamically.
- Language, alt_value, language dependency should remain same for the old and new sublexer preference.
- For updating the default sublexer, the syntax is:

```
UPDATE SUB LEXER DEFAULT TO SUB LEXER PREFERENCE
```

ADD MDATA SECTION secname TAG sectag READ ONLY

The following conditions apply:

- Allows users to add MDATA section to the index.
- Cannot be used with NULL/AUTO/PATH section groups.

ALTER INDEX Examples

Resuming Failed Index

The following statement resumes the indexing operation on newsindex with 2 megabytes of memory:

```
ALTER INDEX newsindex REBUILD PARAMETERS ('resume memory 2M');
```

Rebuilding an Index

The following statement rebuilds the index, replacing the stoplist preference with new stop.

```
ALTER INDEX newsindex REBUILD PARAMETERS('replace stoplist new_stop');
```

Rebuilding a Partitioned Index

The following example creates a partitioned text table, populates it, and creates a partitioned index. It then adds a new partition to the table and rebuilds the index with ALTER INDEX as follows:

```
PROMPT create partitioned table and populate it
create table part tab (a int, b varchar2(40)) partition by range(a)
(partition p tab1 values less than (10),
partition p tab2 values less than (20),
partition p tab3 values less than (30));
insert into part tab values (1, 'Actinidia deliciosa');
insert into part tab values (8, 'Distictis buccinatoria');
insert into part tab values (12, 'Actinidia quinata');
insert into part tab values (18, 'Distictis Rivers');
insert into part tab values (21, 'pandorea jasminoides');
insert into part tab values (28, 'pandorea rosea');
commit:
PROMPT create partitioned index
create index part idx on part tab(b) indextype is ctxsys.context
local (partition p idx1, partition p idx2, partition p idx3);
PROMPT add a partition and populate it
alter table part tab add partition p tab4 values less than (40);
insert into part tab values (32, 'passiflora citrina');
```



```
insert into part_tab values (33, 'passiflora alatocaerulea');
commit;
```

The following statement rebuilds the index in the newly populated partition. In general, the index partition name for a newly added partition is the same as the table partition name, unless the name has already been used. In this case, Oracle Text generates a new name.

```
alter index part_idx rebuild partition p_tab4;
```

The following statement queries the table for the two hits in the newly added partition:

```
select * from part tab where contains(b,'passiflora') >0;
```

The following statement queries the newly added partition directly:

```
select * from part tab partition (p tab4) where contains(b,'passiflora') >;
```

Replacing Index Metadata: Changing Single-Lexer to Multilexer

The following example demonstrates how an application can migrate from single-language documents (English) to multilanguage documents (English and Spanish) by replacing the index metadata for the lexer.

```
REM creates a simple table, which stores only English (American) text
create table simple (text varchar2(80));
insert into simple values ('the quick brown fox');
commit:
REM create a simple lexer to lex this English text
  ctx ddl.create preference('us lexer', 'basic lexer');
end;
REM create a text index on the simple table
create index simple idx on simple(text)
indextype is ctxsys.context parameters ('lexer us_lexer');
REM we can query easily
select * from simple where contains(text, 'fox')>0;
REM now suppose we want to start accepting Spanish documents.
REM first we have to extend the table with a language column
alter table simple add (lang varchar2(10) default 'us');
REM now let's create a Spanish lexer,
begin
  ctx ddl.create preference('e lexer', 'basic lexer');
  ctx ddl.set attribute('e lexer', 'base letter', 'yes');
end;
REM Then create a multilexer incorporating our English and Spanish lexers.
REM Note that the DEFAULT lexer is the exact same lexer, with which we have
REM have already indexed all the documents.
  ctx ddl.create preference('m lexer', 'multi lexer');
  ctx ddl.add sub lexer('m lexer', 'default', 'us lexer');
  ctx ddl.add sub lexer('m lexer', 'spanish', 'e lexer');
end;
REM next replace our metadata
```



```
alter index simple_idx rebuild parameters ('replace metadata language column lang lexer m_lexer');

REM We are ready for some Spanish data. Note that we could have inserted REM this BEFORE the alter index, as long as we did not SYNC. insert into simple values ('el zorro marr@oacute;n r@aacute;pido', 'e'); commit; exec ctx_ddl.sync_index('simple_idx');

REM now query the Spanish data with base lettering: select * from simple where contains(text, 'rapido')>0;
```

Optimizing the Index

To optimize your index, use CTX_DDL.OPTIMIZE_INDEX.

Synchronizing the Index

To synchronize your index, use CTX_DDL.SYNC_INDEX.

Adding a Zone Section

To add to the index the zone section author identified by the tag <author>, enter the following statement:

ALTER INDEX myindex REBUILD PARAMETERS('add zone section author tag author');

Adding a Stop Section

To add a stop section identified by tag <fluff> to the index that uses the AUTO_SECTION_GROUP, enter the following statement:

```
ALTER INDEX myindex REBUILD PARAMETERS ('add stop section fluff');
```

Adding an Attribute Section

Assume that the following text appears in an XML document:

```
<book title="Tale of Two Cities">It was the best of times.
```

Assume also that you want to create a separate section for the title attribute and you want to name the new attribute section booktitle. To do so, enter the following statement:

```
ALTER INDEX myindex REBUILD PARAMETERS('add attr section booktitle tag title@book');
```

Adding an SDATA Section

To add an SDATA section S1 of NUMBER data type and identified by tag T1, to the index, enter the following statement:

```
ALTER INDEX myindex PARAMETERS ('add sdata section S1 tag T1 datatype NUMBER);
```

Disabling Automatic Background Index Optimization

The following example disables optimize token and optimize full jobs which are automatically running in the background:

```
ALTER INDEX myindex PARAMETERS ('REPLACE METADATA OPTIMIZE (MANUAL)');
```

Using Flashback Queries

If a Text query is flashed back to a point before an ALTER INDEX statement was issued on the Text index for which the query is being run, then:



- The query optimizer will not choose the index access path for that given index because the
 index is treated according to its creation time with ALTER INDEX. Therefore, to the query
 optimizer, the index is perceived not to exist.
- The functional processing of the Text operator will fail with ORA-01466 or ORA-08176 errors if the ALTER INDEX statement involves re-creation of DR\$ index tables.

To work around this issue, use the DBMS FLASHBACK package. For example:

```
EXEC dbms_flashback.enable_at_system_change_number(:scn);
SELECT id from documents WHERE CONTAINS(text, 'oracle')>0;
EXEC dbms flashback.disable;
```



Using DBMS_FLASHBACK Package in Oracle Database Development Guide

Notes

Add Section Constraints

Before altering the index section information, Oracle Text checks the new section against the existing sections to ensure that all validity constraints are met. These constraints are the same for adding a section to a section group with the CTX DDL PL/SQL package and are as follows:

- You cannot add zone, field, or stop sections to a NULL_SECTION_GROUP.
- You cannot add zone, field, or attribute sections to an automatic section group.
- You cannot add attribute sections to anything other than XML section groups.
- You cannot have the same tag for two different sections.
- Section names for zone, field, and attribute sections cannot intersect.
- You cannot exceed 64 fields per section.
- You cannot add stop sections to basic, HTML, XML, or news section groups.
- SENTENCE and PARAGRAPH are reserved section names.
- You cannot have embedded blanks in section and field names.

Related Topics

- Oracle Text Indexing Elements
- CREATE INDEX

Use the CREATE INDEX statement to create an Oracle Text index.

ALTER INDEX

Use the ALTER INDEX statement to change or rebuild an existing index, such as Oracle Text index, Oracle Text search index, JSON search index, XML search index, or hybrid vector index.

OPTIMIZE INDEX

Use this procedure to optimize the index. Optimizing an index removes old data and minimizes index fragmentation, which can improve query response time.



1.2 ALTER TABLE: Supported Partitioning Statements



This section describes the ALTER TABLE statement as it pertains to adding and modifying a partitioned text table with a context domain index.

For a complete description of the ALTER TABLE statement, see *Oracle Database SQL Language Reference*.

Purpose

Use the ALTER TABLE statement to add, modify, split, merge, exchange, or drop a partitioned text table with a context domain index. The following sections describe some of the ALTER TABLE operations.

Modify Partition Syntax

Unusable Local Indexes

ALTER TABLE [schema.]table MODIFY PARTITION partition UNUSABLE LOCAL INDEXES

Marks the index partition corresponding to the given table partition UNUSABLE. You might mark an index partition unusable before you rebuild the index partition as described in "Rebuild Unusable Local Indexes".

If the index partition is not marked unusable, then the statement returns without actually rebuilding the local index partition.

Rebuild Unusable Local Indexes

ALTER TABLE [schema.]table MODIFY PARTITION partition REBUILD UNUSABLE LOCAL INDEXES

Rebuilds the index partition corresponding to the specified table partition that has an UNUSABLE status.



If the index partition status is already VALID before you enter this statement, then this statement does *not* rebuild the index partition. Do not depend on this statement to rebuild the index partition unless the index partition status is UNUSABLE.

Add Partition Syntax

ALTER TABLE [schema.]table ADD PARTITION [partition] VALUES LESS THAN (value list) [partition description]

Adds a new partition to the high end of a range-partitioned table.



To add a partition to the beginning or to the middle of the table, use the ALTER TABLE SPLIT PARTITION statement.

The newly added table partition is always empty, and the context domain index (if any) status for this partition is always VALID. After issuing DML, if you want to synchronize or optimize this newly added index partition, then you must look up the index partition name and enter the ALTER INDEX REBUILD PARTITION statement. For this newly added partition, the index partition name is usually the same as the table partition name, but if the table partition name is already used by another index partition, the system assigns a name in the form of SYS Pn.

By querying the <code>USER_IND_PARTITIONS</code> view and comparing the <code>HIGH_VALUE</code> field, you can determine the index partition name for the newly added partition.

Merge Partition Syntax

```
ALTER TABLE [schema.]table
MERGE PARTITIONS partition1, partition2
[INTO PARTITION [new_partition] [partition_description]]
[UPDATE GLOBAL INDEXES]
```

Applies only to a range partition. This statement merges the contents of two adjacent partitions into a new partition and then drops the original two partitions. If the resulting partition is non-empty, then the corresponding local domain index partition is marked UNUSABLE. You can use ALTER TABLE MODIFY PARTITION to rebuild the partition index.

Note:

For a global, nonpartitioned index, if you perform the merge operation without an <code>UPDATE GLOBAL INDEXES</code> clause, then the resulting index (if not <code>NULL</code>) will be invalid and must be rebuilt. If you specify the <code>UPDATE GLOBAL INDEXES</code> clause after the operation and the <code>SYNC</code> type is <code>MANUAL</code>, then the index will be valid, but you still must synchronize the index with <code>CTX_DDL.SYNC_INDEX</code> for the update to take place.

The naming convention for the resulting index partition is the same as in the ALTER TABLE ADD PARTITION statement.

Split Partition Syntax

```
ALTER TABLE [schema.]table

SPLIT PARTITION partition_name_old

AT (value_list)
[into (partition_description, partition_description)]
[parallel_clause]
[UPDATE GLOBAL INDEXES]
```

Applies only to range partitions. This statement divides a table partition into two partitions, thus adding a new partition to the table. The local corresponding index partitions will be marked UNUSABLE if the corresponding table partitions are non-empty. Use the ALTER TABLE MODIFY PARTITION statement to rebuild the partition indexes.



For a global, nonpartitioned index, if you perform the split operation without an UPDATE GLOBAL INDEXES clause, then the resulting index (if not NULL) will be invalid and must be rebuilt. If you specify the UPDATE GLOBAL INDEXES clause after the operation and the SYNC type is MANUAL, then the index will be valid, but you still must synchronize the index with CTX_DDL.SYNC_INDEX for the update to take place.

The naming convention for the two resulting index partition is the same as in the ALTER TABLE ADD PARTITION statement.

Exchange Partition Syntax

ALTER TABLE [schema.]table EXCHANGE PARTITION partition WITH TABLE table [INCLUDING|EXCLUDING INDEXES} [WITH|WITHOUT VALIDATION] [EXCEPTIONS INTO [schema.]table] [UPDATE GLOBAL INDEXES]

Converts a partition to a nonpartitioned table, and converts a table to a partition of a partitioned table by exchanging their data segments. Rowids are preserved.

If EXCLUDING INDEXES is specified, all the context indexes corresponding to the partition and all the indexes on the exchanged table are marked as UNUSABLE. To rebuild the new index partition in this case, issue an ALTER TABLE MODIFY PARTITION statement.

If INCLUDING INDEXES is specified, then for every local domain index on the partitioned table, there must be a nonpartitioned domain index on the nonpartitioned table. The local index partitions are exchanged with the corresponding regular indexes.

Note:

For a global, nonpartitioned index, if you perform the exchange operation without an UPDATE GLOBAL INDEXES clause, then the resulting index (if not NULL) will be invalid and must be rebuilt. If you specify the UPDATE GLOBAL INDEXES clause after the operation and the SYNC type is MANUAL, then the index will be valid, but you still must synchronize the index with CTX_DDL.SYNC_INDEX for the update to take place.

Field Sections

Field section queries might not work the same way if the nonpartitioned index and local index use different section IDs for the same field section.

Storage

Storage is not changed. So if the index on the nonpartitioned table \$I table was in tablespace XYZ, then after the exchange partition, it will still be in tablespace XYZ, but now it is the \$I table for an index partition.

Storage preferences are not switched, so if you switch and then rebuild the index, then the table may be created in a different location.

Restrictions



Both indexes must be equivalent. They must use the same objects and the same settings for each object. Note that Oracle Text checks only that the indexes are using the same object. But they should use the same exact everything.

No index object can be partitioned, that is, when the user has used the storage object to partition the \$I, \$N tables.

If either index or index partition does not meet all these restrictions an error is raised and both the index and index partition will be INVALID. You must manually rebuild both index and index partition using the ALTER INDEX REBUILD statement.

Truncate Partition Syntax

```
ALTER TABLE [schema.]table TRUNCATE PARTITION [DROP|REUSE STORAGE] [UPDATE GLOBAL INDEXES]
```

Removes all rows from a partition in a table. Corresponding CONTEXT index partitions are also removed.



For a global, nonpartitioned index, if you perform the truncate operation without an <code>UPDATE GLOBAL INDEXES</code> clause, then the resulting index (if not <code>NULL</code>) will be invalid and must be rebuilt. If you specify the <code>UPDATE GLOBAL INDEXES</code> clause after the operation, the index will be valid.

ALTER TABLE Examples

Global Index on Partitioned Table Examples

The following example creates a range-partitioned table with three partitions. Each partition is populated with two rows. A global, nonpartitioned CONTEXT index is then created. To demonstrate the UPDATE GLOBAL INDEXES clause, the partitions are split and merged with an index synchronization.

```
create table tdrexglb part(a int, b varchar2(40)) partition by range(a)
(partition p1 values less than (10),
partition p2 values less than (20),
partition p3 values less than (30));
insert into tdrexglb part values (1,'row1');
insert into tdrexglb_part values (8,'row2');
insert into tdrexglb_part values (11,'row11');
insert into tdrexglb part values (18,'row18');
insert into tdrexglb part values (21, 'row21');
insert into tdrexglb part values (28, 'row28');
commit;
create index tdrexglb parti on tdrexglb part(b) indextype is ctxsys.context;
create table tdrexglb(a int, b varchar2(40));
insert into tdrexglb values(20, 'newrow20');
commit;
PROMPT make sure query works
select * from tdrexglb part where contains(b,'row18') >0;
```

```
PROMPT split partition
alter table tdrexglb_part split partition p2 at (15) into
(partition p21, partition p22) update global indexes;
PROMPT before sync
select * from tdrexglb part where contains(b,'row11') >0;
select * from tdrexglb part where contains(b,'row18') >0;
exec ctx_ddl.sync_index('tdrexglb_parti')
PROMPT after sync
select * from tdrexglb_part where contains(b,'row11') >0;
select * from tdrexglb part where contains(b,'row18') >0;
PROMPT merge partition
alter table tdrexglb part merge partitions p22, p3
into partition pnew3 update global indexes;
PROMPT before sync
select * from tdrexglb part where contains(b,'row18') >0;
select * from tdrexglb part where contains(b,'row28') >0;
exec ctx ddl.sync index('tdrexglb parti');
PROMPT after sync
select * from tdrexglb part where contains(b,'row18') >0;
select * from tdrexglb_part where contains(b,'row28') >0;
PROMPT drop partition
alter table tdrexglb part drop partition p1 update global indexes;
PROMPT before sync
select * from tdrexglb part where contains(b,'row1') >0;
exec ctx ddl.sync index('tdrexglb parti');
PROMPT after sync
select * from tdrexglb part where contains(b,'row1') >0;
PROMPT exchange partition
alter table tdrexglb part exchange partition pnew3 with table
tdrexglb update global indexes;
PROMPT before sync
select * from tdrexglb part where contains(b,'newrow20') >0;
select * from tdrexglb part where contains(b,'row28') >0;
exec ctx ddl.sync index('tdrexglb parti');
PROMPT after sync
select * from tdrexglb part where contains(b,'newrow20') >0;
select * from tdrexglb part where contains(b,'row28') >0;
PROMPT move table partition
alter table tdrexglb part move partition p21 update global indexes;
PROMPT before sync
select * from tdrexglb part where contains(b,'row11') >0;
exec ctx ddl.sync index('tdrexglb parti');
PROMPT after sync
select * from tdrexglb part where contains(b,'row11') >0;
PROMPT truncate table partition
alter table tdrexglb part truncate partition p21 update global indexes;
```

update global indexes;

1.3 CATSEARCH

Use the Catsearch operator to search CTXCAT indexes. Use this operator in the WHERE clause of a Select statement.

The CATSEARCH operator also supports database links. You can identify a remote table or materialized view by appending <code>@dblink</code> to the end of its name. The <code>dblink</code> must be a complete or partial name for a database link to the database containing the remote table or materialized view. (Indexing of remote views is not supported.)

The grammar of this operator is called CTXCAT. You can also use the CONTEXT grammar if your search criteria require special functionality, such as thesaurus, fuzzy matching, proximity searching, or stemming. To utilize the CONTEXT grammar, use the "Query Template Specification" in the text guery parameter as described in this section.

Note:

The Oracle Text indextype CTXCAT is deprecated with Oracle Database 23ai. The indextype itself, and it's operator CTXCAT, can be removed in a future release. Both CTXCAT and the use of CTXCAT grammar as an alternative grammar for CONTEXT queries is deprecated. Instead, Oracle recommends that you use the CONTEXT indextype, which can provide all the same functionality, except that it is not transactional. Near-transactional behavior in CONTEXT can be achieved by using SYNC (ON COMMIT) or, preferably, SYNC (EVERY [time-period]) with a short time period.

CTXCAT was introduced when indexes were typically a few megabytes in size. Modern, large indexes, can be difficult to manage with CTXCAT. The addition of index sets to CTXCAT can be achieved more effectively by the use of FILTER BY and ORDER BY columns, or SDATA, or both, in the CONTEXT indextype. CTXCAT is therefore rarely an appropriate choice. Oracle recommends that you choose the more efficient CONTEXT indextype.

About Performance

Use the CATSEARCH operator with a CTXCAT index mainly to improve mixed-query performance. Specify your text query condition with text_query and your structured condition with the structured query argument.

Internally, Oracle Text uses a combined B-tree index on text and structured columns to quickly produce results satisfying the query.

Limitations

If the optimizer chooses to use the functional query invocation, then your query will fail. The optimizer might choose functional invocation when your structured clause is highly selective.

You can use the INDEX hint to specify the optimizer to use the index and avoid functional evaluation of CATSEARCH.



The structured_query argument of the CATSEARCH operator must reference columns used during CREATE INDEX sets; otherwise, error DRG-10845 will be raised. For example, the error will be raised if you issue a CATSEARCH query on a view created on top of a table with the CTXCAT index on it, and the name of the logical column on the view is different from the actual column name on the physical table. The columns referenced by the structured_query argument of the CATSEARCH operator must be the physical column name used during CREATE INDEX sets, not the logical column on the view.

Syntax

```
CATSEARCH (
```

```
[schema.]column,
text_query [VARCHAR2|CLOB],
structured_query VARCHAR2,
```

RETURN NUMBER;

[schema.]column

Specifies the text column to be searched on. This column must have a CTXCAT index associated with it.

text_query

Specify one of the following to define your search in column:

- CATSEARCH Query Operations
- Query Template Specification (for using CONTEXT grammar)

CATSEARCH Query Operations

The CATSEARCH operator supports only the following query operations:

- Logical AND
- Logical OR (|)
- Logical NOT (-)
- " " (quoted phrases)
- Wildcarding

CATSEARCH Query Operations provides the syntax for these operators.

Table 1-6 CATSEARCH Query Operators

Operation	Syntax	Description of Operation	
Logical AND	abc	Returns rows that contain a, b, and c.	
Logical OR	a b c	Returns rows that contain a, b, or c.	
Logical NOT	a - b	Returns rows that contain a and not b.	
Hyphen with no	a-b	Hyphen treated as a regular character.	
space		For example, if the hyphen is defined as skipjoin, words such as web-site are treated as the single query term website.	
		Likewise, if the hyphen is defined as a printjoin, words such as <i>web-site</i> are treated as <i>web-site</i> in the CTXCAT query	
		language.	



Table 1-6 (Cont.) CATSEARCH Query Operators

Operation	Syntax	Description of Operation	
11 11	"a b c"	Returns rows that contain the phrase "a b c".	
		For example, entering "Sony CD Player" means return all rows that contain this sequence of words.	
()	(A B) C	Parentheses group operations. This query is equivalent to the CONTAINS query (A &B) \mid C.	
Wildcard	term*	The wildcard character matches zero or more characters.	
(right and double truncated)	a*b	For example, do* matches dog, and gl*s matches glass.	
		Left truncation not supported.	
		Note: Oracle recommends that you create a prefix index if your application uses wildcard searching. Set prefix indexing with the BASIC_WORDLIST preference.	

The following limitations apply to these operators:

- The left-hand side (the column name) must be a column named in at least one of the indexes of the index set.
- The left-hand side must be a plain column name. Functions and expressions are not allowed.
- The right-hand side must be composed of literal values. Functions, expressions, other columns, and subselects are not allowed.
- Multiple criteria can be combined with AND. Note that OR is not supported.
- When querying a remote table through a database link, the database link must be specified for CATSEARCH as well as for the table being queried.

For example, these expressions are supported:

```
catsearch(text, 'dog', 'foo > 15')
catsearch(text, 'dog', 'bar = ''SMITH''')
catsearch(text, 'dog', 'foo between 1 and 15')
catsearch(text, 'dog', 'foo = 1 and abc = 123')
catsearch@remote(text, 'dog', 'foo = 1 and abc = 123')
```

These expressions are not supported:

```
catsearch(text, 'dog', 'upper(bar) = ''A''')
catsearch(text, 'dog', 'bar LIKE ''A%''')
catsearch(text, 'dog', 'foo = abc')
catsearch(text, 'dog', 'foo = 1 or abc = 3')
```

Query Template Specification

Specifies a marked-up string that specifies a query template. Specify one of the following templates:

- Query rewrite, used to expand a query string into different versions
- Progressive relaxation, used to progressively enter less restrictive versions of a query to increase recall
- Alternate grammar, used to specify CONTAINS operators (See "CONTEXT Query Grammar Examples")
- Alternate language, used to specify alternate query language



Alternate scoring, used to specify alternate scoring algorithms



The text_query parameter description for CONTAINS for more information about the syntax for these query templates

structured_query

Specifies the structured conditions and the ORDER BY clause. There must exist an index for any column you specify. For example, if you specify 'category_id=1 order by bid_close', you must have an index for 'category_id, bid_close' as specified with the CTX_DDL.ADD_INDEX package.

With structured query, you can use standard SQL syntax only with the following operators:

- =
- <=
- >=
- >
- <
- IN
- BETWEEN
- AND (to combine two or more clauses)



You cannot use parentheses () in the structured query parameter.

Examples

Create the table.

The following statement creates the table to be indexed:

```
CREATE TABLE auction (category_id number primary key, title varchar2(20), bid close date);
```

The following statements insert the values into the table:

```
INSERT INTO auction values(1, 'Sony DVD Player', '20-FEB-2012');
INSERT INTO auction values(2, 'Sony DVD Player', '24-FEB-2012');
INSERT INTO auction values(3, 'Pioneer DVD Player', '25-FEB-2012');
INSERT INTO auction values(4, 'Sony DVD Player', '25-FEB-2012');
INSERT INTO auction values(5, 'Bose Speaker', '22-FEB-2012');
INSERT INTO auction values(6, 'Tascam CD Burner', '25-FEB-2012');
INSERT INTO auction values(7, 'Nikon digital camera', '22-FEB-2012');
INSERT INTO auction values(8, 'Canon digital camera', '26-FEB-2012');
```

2. Create the CTXCAT index.

The following statements create the CTXCAT index:

begin



```
ctx_ddl.create_index_set('auction_iset');
ctx_ddl.add_index('auction_iset','bid_close');
end;
/
CREATE INDEX auction_titlex ON auction(title) INDEXTYPE IS CTXSYS.CTXCAT
PARAMETERS ('index set auction iset');
```

3. Query the table.

A typical query with CATSEARCH might include a structured clause as follows to find all rows that contain the word *camera* ordered by bid close:

The following query finds all rows that contain the phrase *Sony DVD Player* and that have a bid close date of February 20, 2012:

The following query finds all rows with the terms Sony and DVD and Player:

The following query finds all rows with the term *DVD* and not *Player*:

6 Tascam CD Burner 25-FEB-12

The following query finds all rows with the terms *CD* or *DVD* or *Speaker*:

```
SELECT * FROM auction WHERE CATSEARCH(title, 'CD | DVD | Speaker', 'order by bid close desc')> 0;
```

```
CATEGORY_ID TITLE BID_CLOSE

3 Pioneer DVD Player 25-FEB-12
4 Sony DVD Player 25-FEB-12
6 Tascam CD Burner 25-FEB-12
2 Sony DVD Player 24-FEB-12
5 Bose Speaker 22-FEB-12
1 Sony DVD Player 20-FEB-12
```



The following query finds all rows that are about audio equipment:

```
SELECT * FROM auction WHERE CATSEARCH(title, 'ABOUT(audio equipment)', NULL) > 0;
```

CONTEXT Query Grammar Examples

The following examples show how to specify the CONTEXT grammar in CATSEARCH queries using the template feature:

```
PROMPT
PROMPT fuzzy: query = ?test
PROMPT should match all fuzzy variations of test (for example, text)
select pk||' ==> '||text from test
where catsearch(text,
'<query>
 <textquery grammar="context">
     ?test
  </textquery>
</query>','')>0
order by pk;
PROMPT
PROMPT fuzzy: query = !sail
PROMPT should match all soundex variations of bot (for example, sell)
select pk||' ==> '||text from test
where catsearch (text,
'<query>
  <textquery grammar="context">
     !sail
 </textquery>
</query>','')>0
order by pk;
PROMPT
PROMPT theme (ABOUT) query
PROMPT query: about (California)
select pk||' ==> '||text from test
where catsearch (text,
'<query>
 <textquery grammar="context">
     about (California)
 </textquery>
</query>','')>0
order by pk;
```

The following example shows a field section search against a CTXCAT index using CONTEXT grammar by means of a query template in a CATSEARCH query:

```
-- Create and populate table
create table BOOKS (ID number, INFO varchar2(200), PUBDATE DATE);

insert into BOOKS values(1, '<author>NOAM CHOMSKY</author><subject>CIVIL
   RIGHTS</subject><language>ENGLISH</language><publisher>MIT
   PRESS</publisher>', '01-NOV-2003');

insert into BOOKS values(2, '<author>NICANOR PARRA</author><subject>POEMS
   AND ANTIPOEMS</subject><language>SPANISH</language>
   <publisher>VASQUEZ</publisher>', '01-JAN-2001');

insert into BOOKS values(1, '<author>LUC SANTE</author><subject>XML
   DATABASE</subject><language>FRENCH</language><publisher>FREE
   PRESS</publisher>', '15-MAY-2002');
```



```
commit:
-- Create index set and section group
exec ctx ddl.create index set('BOOK INDEX SET');
exec ctx ddl.add index('BOOKSET','PUBDATE');
exec ctx ddl.create section group('BOOK SECTION GROUP',
      'BASIC SECTION GROUP');
exec ctx ddl.add field section('BOOK SECTION GROUP', 'AUTHOR', 'AUTHOR');
exec ctx_ddl.add_field_section('BOOK_SECTION_GROUP','SUBJECT','SUBJECT');
exec ctx_ddl.add_field_section('BOOK_SECTION_GROUP','LANGUAGE');
exec ctx ddl.add field section('BOOK SECTION GROUP', 'PUBLISHER', 'PUBLISHER');
-- Create index
create index books index on books(info) indextype is ctxsys.ctxcat
 parameters ('index set book index set section group book section group');
-- Use the index
-- Note that: even though CTXCAT index can be created with field sections, it
-- cannot be accessed using CTXCAT grammar (default for CATSEARCH).
-- We need to use query template with CONTEXT grammar to access field
-- sections with CATSEARCH.
select id, info from books
where catsearch (info,
'<query>
      <textquery grammar="context">
            NOAM within author and english within language
      </textquery>
</query>',
'order by pubdate')>0;
```

Related Topics

"Syntax for CTXCAT Index Type"

Oracle Text Application Developer's Guide

1.4 CONTAINS

Use the CONTAINS operator in the WHERE clause of a SELECT statement to specify the query expression for a Text query.

The CONTAINS operator also supports database links. You can identify a remote table or materialized view by appending <code>@dblink</code> to the end of its name. The <code>dblink</code> must be a complete or partial name for a database link to the database containing the remote table or materialized view (querying of remote views is not supported).

CONTAINS returns a relevance score for every row selected. Obtain this score with the SCORE operator.

The grammar for this operator is called the CONTEXT grammar. You can also use CTXCAT grammar if your application works better with simpler syntax. To do so, use the Query Template Specification in the text_query parameter as described in this section.



The Oracle Text indextype CTXCAT is deprecated with Oracle Database 23ai. The indextype itself, and it's operator CTXCAT, can be removed in a future release. Both CTXCAT and the use of CTXCAT grammar as an alternative grammar for CONTEXT queries is deprecated. Instead, Oracle recommends that you use the CONTEXT indextype, which can provide all the same functionality, except that it is not transactional. Near-transactional behavior in CONTEXT can be achieved by using SYNC (ON COMMIT) or, preferably, SYNC (EVERY [time-period]) with a short time period.

CTXCAT was introduced when indexes were typically a few megabytes in size. Modern, large indexes, can be difficult to manage with CTXCAT. The addition of index sets to CTXCAT can be achieved more effectively by the use of FILTER BY and ORDER BY columns, or SDATA, or both, in the CONTEXT indextype. CTXCAT is therefore rarely an appropriate choice. Oracle recommends that you choose the more efficient CONTEXT indextype.

See Also:

- Query Rewrite Template
- Query Result Set Descriptor Template
- Query Relaxation Template
- Alternate Grammar Template
- Language Independent Template
- Alternate Language Template
- Alternative Scoring Template
- The CONTEXT Grammar" topic in Oracle Text Application Developer's Guide

Syntax

[schema.]column

Specify the text column to be searched on. This column must have a Text index associated with it.

text query

Specify one of the following (limited to 4000 bytes for a VARCHAR2 or 64000 bytes for a CLOB):

- The query expression that defines your search in column.
- A marked-up document that specifies a query template.

Use one of the following query templates:



- Query Rewrite Template
- Query Result Set Descriptor Template
- Query Relaxation Template
- Alternate Grammar Template
- Language Independent Template
- Alternate Language Template
- Alternative Scoring Template

Query Rewrite Template

Use this template to automatically write different versions of a query before you submit the query to Oracle Text. This is useful when you need to maximize the recall of a user query. For example, you can program your application to expand a single phrase query of 'cat dog' into the following queries:

```
{cat} {dog}
{cat} ; {dog}
{cat} AND {dog}
{cat} ACCUM {dog}
```

These queries are submitted as one query and results are returned with no duplication. In this example, the query returns documents that contain the phrase *cat dog* as well as documents in which *cat* is near *dog*, and documents that have *cat* and *dog*.

This is done with the following template:

The operator TRANSFORM is used to specify the rewrite rules and has the following syntax (note that it uses double parentheses). The parameters are described in the following table.

```
TRANSFORM((terms, prefix, suffix, connector))
```

Table 1-7 TRANSFORM Parameters

Parameter	Description	
term	Specifies the type of terms to be produced from the original query. Specify either TOKENS or THEMES.	
prefix	Specifies the literal string to be prepended to all terms.	
suffix	Specifies the literal string to be appended to all terms.	
connector	Specifies the literal string to connect all terms after applying the prefix and suffix.	



An error will be raised if the input Text query string specified in the Query Rewrite Template with TRANSFORM rules contains any Oracle Text query operators (such as AND, OR, or SOUNDEX). Also, any special characters (such as % or %) in the input Text query string must be preceded by an escape character, or an error is raised.

Query Result Set Descriptor Template

Use this template to take in a Result Set Descriptor. The element ctx_result_set_descriptor is added to the query template. This enables the CONTAINS query cursor to take in a group count query.

The Result Set Interface document is placed in a public variable in the ctx_query package. (ctx_query.result_set_document.)

The CONTAINS query cursor behavior remains unchanged and the Result Set Document is available right after closing the cursor

For example, the following query of kukui nut returns a result set with the following template.

```
<query>
                       <textquery lang="ENGLISH" grammar="CONTEXT">
                                     cprogression>
                                                     \label{lem:condition} $$ \enskip < \enskip <
                                                    \label{lem:condition} $$ \ensuremath{$<$}$ \ensuremath{$<$$}$ \ensuremath{$<$$}$ \ensuremath{$<$$}$ \ensuremath{$<$}$ \ensuremath{$<$$}$ \ensuremath{$<$$}
                                                     <seq><rewrite>transform((TOKENS, "{", "}", " AND "))</rewrite></seq>
                                                     <seq><rewrite>transform((TOKENS, "{", "}", " ACCUM "))</rewrite></seq>
                                     </textquery>
                <score datatype="INTEGER" algorithm="COUNT"/>
<ctx result set descriptor>
       <group>
                               <qroup values>
                                                  <value id="2"/>
                                                   <value id="3"/>
                                                     <value id="4"/>
                              </group values>
                               <count/>
               </group>
</ctx result set descriptor>
</query>
```

Query Relaxation Template

Use this template to progressively relax your query. **Progressive relaxation** is when you increase recall by progressively issuing less restrictive versions of a query, so that your application can return an appropriate number of hits to the user.

For example, the query of *blue pen* can be progressively relaxed to:

```
blue pen
blue NEAR pen
blue AND pen
blue ACCUM pen
```

This is done with the following template



```
<query>
    <textquery lang="ENGLISH" grammar="CONTEXT">
        <progression>
            <seq>blue pen</seq>
            <seq>blue NEAR pen</seq>
            <seq>blue AND pen</seq>
            <seq>blue ACCUM pen</seq>
            <progression>
            </textquery>
            <score datatype="INTEGER" algorithm="COUNT"/>
</query>
```

Alternate Grammar Template

Use this template to specify an alternate grammar, such as CONTEXT or CATSEARCH. Specifying an alternate grammar enables you to enter queries using different syntax and operators.

For example, with CATSEARCH, enter ABOUT queries using the CONTEXT grammar. Likewise with CONTAINS, enter logical queries using the simplified CATSEARCH syntax.

The phrase 'dog cat mouse' is interpreted as a phrase in CONTAINS. However, with CATSEARCH, this is equivalent to an AND query of 'dog AND cat AND mouse'. Specify that CONTAINS use the alternate grammar with the following template:

```
<query>
    <textquery grammar="CTXCAT">dog cat mouse</textquery>
    <score datatype="integer"/>
    </query>
```

Language Independent Template

Use this template to specify a lexer that uses user-defined symbols (or abbreviations) and does not depend on any language.

The following example specifies that the query take a list of language-independent sublexers.

The following conditions apply:

- The sublexers element consists of one or more sublexer label elements.
- Each sublexer_label element contains the symbol for the language independent sub lexer.
- When the sublexers element is specified, the query will be processed with the stopwords
 and sub_lexers for each of the symbols specified in the sublexers element, and query will
 return only the documents indexed by the specified sub_lexers.
- A special reserved symbol called SESSION_LANG can be used for the system to pick a language-dependent sub_lexer based on the language specified in lang attribute of the textguery element in the guery template. If lang attribute is not specified, then the language specified in language specified, then the language specified is not specified.



attribute will be based on session language. Query parsed by the chosen sub_lexer will only return documents indexed by that language-dependent sub_lexer. If both ${\tt SESSION_LANG}$ and ${\tt lang}$ attribute are specified, the ${\tt lang}$ attribute will take priority.

- If sublexers element is specified without SESSION_LANG, then lang attribute of textquery element will be ignored.
- Default Behavior:

If sublexers element is not present in the query template, then query will be parsed with one language-dependent sub-lexer (if any), which is chosen based on the specified lang attribute value or the session language AND all language independent sub-lexers.

Alternate Language Template

Use this template to specify an alternate language:

```
<query><textquery lang="french">bon soir</textquery></query>
```

Alternative Scoring Template

Use this template to specify an alternative scoring algorithm.

The following example specifies that the query use the CONTEXT grammar and return integer scores using the COUNT algorithm. This algorithm returns a score as the number of query occurrences in the document.

The following example uses the normalization_expr attribute to add SDATA(price) into the score returned by the query, and uses it as the final score:

The $normalization_expr$ attribute is used only with the alternate scoring template, and is an arithmetic expression that consists of:

- Arithmetic operators: + * /. The operator precedence is the same as that for SQL operator precedence.
- Grouping operators: (). Parentheses can be used to alter the precedence of the arithmetic operators.
- Absolute function: ABS(n) returns the absolute value of n; where n is any expression that returns a number.
- Logarithmic function: LOG(*n*): returns the base-10 logarithmic value of *n*; where *n* is any expression that returns a number.
- Predefined components: The doc_score predefined component can be used to return the initial query score of a particular document.
- SDATA component: SDATA(name) returns the value of the SDATA with the specified name as the score.



- Only SDATA with a NUMBER or DATE data type is allowed. An error is raised otherwise.
- The sdata string and the SDATA name are case-insensitive.
- Because an SDATA section value can be NULL, any expression with NULL SDATA section value is evaluated as 0. For example: the normalization_expr "doc_score + SDATA(price)" will be evaluated to 0 if SDATA(price) for a given document has a NULL value.
- Numeric literals: There are any number literal that conforms to the SQL pattern of NUMBER literal and is within the range of the double-precision floating-point (-3.4e38 to 3.4e38).
- Date literals: Date literals must be enclosed with DATE (). Only the following format is allowed: YYYY-MM-DD or YYYY-MM-DD HH24:MI:SS. For example: DATE (2005-11-08).

Consistent with SQL, if no time is specified, then 00:00:00 is assumed.

The normalization_expr attribute overrides the algorithm attribute. That is, if algorithm is set to COUNT, and the user also specifies normalization_expr, then the score will not be count, but the calculated score based on the normalization expr.

If the score (either from algorithm = COUNT or normalization_expr = ...) is internally calculated to be greater than 100, then it will be set to 100.

If the query relaxation template is used, the score will be further normalized in such a way that documents returned from higher sequences will always have higher scores than documents returned from sequence(s) below.

DATE Literal Restrictions

Only the minus (-) operator is allowed between date-type data (DATE literals and date-type SDATA). Using other operators will result in an error. Subtracting two date-type data will produce a number (float) that represents the difference in number of days between the two dates. For example, the following expression is allowed:

```
SDATA (dob) - DATE (2005-11-08)
```

The following expression is not allowed:

```
SDATA (dob) + DATE (2005-11-08)
```

The plus (+) and minus (-) operators are allowed between numeric data and date type of data. The number operand is interpreted as the number or fraction of days. For example, the following expression is allowed:

```
DATE (2005-11-08) + 1 = 9 \text{ NOV } 2005
```

The following expression is not allowed:

```
DATE (2005-11-08) * 3 = ERROR
```

Template Attribute Values

Table 1-8 gives the possible values for template attributes.

Table 1-8 Template Attribute Values

Tag Attribute	Description	Possible Values	Meaning
grammar=	Specifies the grammar	CONTEXT	The grammar of the query.
of the query.	of the query.	CTXCAT	



Table 1-8 (Cont.) Template Attribute Values

Tag Attribute	Description	Possible Values	Meaning
datatype=	Specifies the type of number returned as score.	INTEGER	Returns score as integer between 0 and 100. Returns score as its high-precision floating-point number between 0 and 100.
		FLOAT	
algorithm=	Specifies the scoring algorithm to use.	DEFAULT	Returns the default.
		COUNT	Returns scores as the number of occurrences in the document.
lang=	Specifies the language name.	Any language supported by Oracle Database. See Oracle Database Globalization Support Guide.	The language name.

Template Grammar Definition

The query template interface is an XML document. Its grammar is defined with the following XML DTD:

```
<!DOCTYPE query [
<!ELEMENT query (textquery, score?, order?)>
<!ELEMENT textquery (#PCDATA|progression)*>
<!ELEMENT progression (seq)+>
<!ELEMENT seq (#PCDATA|rewrite)*>
<!ELEMENT rewrite (#PCDATA)>
<!ELEMENT score EMPTY>
<!ELEMENT order (orderkey+)>
<!ELEMENT order(orderkey+)>
<!ELEMENT orderkey (#PCDATA)>
<!ATTLIST textquery grammar (CONTEXT | CTXCAT | CTXRULE) #REQUIRED>
<!ATTLIST textquery lang CDATA #IMPLIED>
<!ATTLIST score datatype (integer | float) "integer">
<!ATTLIST score algorithm (default | count) "default">
<!ATTLIST score normalization_expr CDATA >
```

Values are case insensitive: integer | float, default | count, context |ctxcat .



Oracle Text CONTAINS Query Operators for more information about the operators in query expressions

label

Optionally, specifies the label that identifies the score generated by the CONTAINS operator.

Returns

For each row selected, the CONTAINS operator returns a number between 0 and 100 that indicates how relevant the document row is to the query. The number 0 means that Oracle Text found no matches in the row.



You must use the SCORE operator with a label to obtain this number.

Example

The following example searches for all documents in the text column that contain the word *oracle*. The score for each row is selected with the SCORE operator using a label of 1:

```
SELECT SCORE(1), title from newsindex
WHERE CONTAINS(text, 'oracle', 1) > 0;
```

The CONTAINS operator must be followed by an expression such as > 0, which specifies that the score value calculated must be greater than zero for the row to be selected.

When the SCORE operator is called (for example, in a SELECT clause), the CONTAINS clause must reference the score label value as in the following example:

```
SELECT SCORE(1), title from newsindex
WHERE CONTAINS(text, 'oracle', 1) > 0 ORDER BY SCORE(1) DESC;
```

The following example specifies that the query be parsed using the CATSEARCH grammar:

Grammar Template Example

The following example shows how to use the CTXCAT grammar in a CONTAINS query. The example creates a CTXCAT and a CONTEXT index on the same table, and compares the query results.

```
PROMPT create context and ctxcat indexes, both using theme indexing
PROMPT
create index tdrbqcq101x on test(text) indextype is ctxsys.context
parameters ('lexer theme lexer');
create index tdrbqcq101cx on test(text) indextype is ctxsys.ctxcat
parameters ('lexer theme lexer');
PROMPT ***** San Diego
                                   *******
PROMPT **** CONTEXT grammar
PROMPT ** should be interpreted as phrase query **
select pk||' ==> '||text from test
where contains(text, 'San Diego')>0
order by pk;
PROMPT **** San Diego
PROMPT ***** CTXCAT grammar ********
PROMPT ** should be interpreted as AND query ***
select pk||' ==> '||text from test
where contains (text,
  <textquery grammar="CTXCAT">San Diego</textquery>
```



```
<score datatype="integer"/>
</query>')>0
order by pk;

PROMPT ***** Hitlist from CTXCAT index *********
select pk||' ==> '||text from test
where catsearch(text,'San Diego','')>0
order by pk;
```

Alternate Scoring Query Template Example

The following query template adds price SDATA section (or SDATA filter-by column) value into the score returned by the query and uses it as the final score:

Query Relaxation Template Example

The following query template defines a query relaxation sequence. The query of *blue pen* is entered in sequence as *blue pen*, then *blue NEAR pen*, then *blue AND pen*, and then *blue ACCUM pen*. Query hits are returned in this sequence with no duplication as long as the application requires results.

Query relaxation is most effective when your application requires the top n hits to a query, which you can obtain with the <code>DOMAIN_INDEX_SORT</code> or <code>FIRST_ROWS</code> hint, which is being deprecated, in a PL/SQL cursor.

Query Rewrite Template Example

The following template defines a query rewrite sequence. The query of *kukui nut* is rewritten as follows:



Order By SDATA Sections Template Example

The following query template defines a query sequence for ordering by SDATA section values using the <order> and <orderkey> elements. The first level of ordering is done on the SDATA section price, which is sorted in the ascending order. The second and third level of ordering is done by the SDATA section pub_date and score, both of which are sorted in the descending order.

The <orderkey> element value must have the following format:

```
<orderkey> SDATA(sdata_section_name) | score [DESC|ASC] </orderkey>
```

The sort order is ascending by default, if not specified as either DESC or ASC.

The <orderkey> element will be ignored in the following cases:

- when the Oracle Cost-Based Optimizer (CBO) pushes the SQL query level ordering into the Text index
- when the CONTAINS() predicate is processed functionally
- when the ordering is already specified by the ORDER BY clause in the SQL query statement

Notes

Ouerving Multilanguage Tables

With the multilexer preference, you can create indexes from multilanguage tables. At query time, the multilexer examines the session's language setting and uses the sublexer preference for that language to parse the query. If the language setting is not mapped, then the default lexer is used.

When the language setting is mapped, the query is parsed and run as usual. The index contains tokens from multiple languages, so such a query can return documents in several languages.

To limit your query to returning documents of a given language, use a structured clause on the language column.

Query Performance Limitation with a Partitioned Index

Oracle Text supports the CONTEXT indexing and querying of a partitioned text table.



However, for optimal performance when querying a partitioned table with an ORDER BY SCORE clause, query the partition. If you query the entire table and use an ORDER BY SCORE clause, the query might not perform optimally unless you include a range predicate that can limit the query to a single partition.

For example, the following statement queries the partition p tab4 partition directly:

```
select * from part_tab partition (p_tab4) where contains(b,'oracle') > 0 ORDER BY
SCORE DESC;
```

Limitation with Remote Execution of CONTAINS Query

Oracle Text supports the remote execution of the CONTAINS operator, but with some limitations. You can invoke the CONTAINS operator in a remote query only if the query is executed completely in the remote database. You cannot use the CONTAINS operator in a subquery of a query, which causes the query to run partly on the remote database and partly on the local database. Doing so will raise the error "ORA-00949: illegal reference to remote database." However, CONTAINS, when invoked remotely from an inner query might run successfully sometimes if view merging is enabled and possible on this query, as in this case the query will be transformed into a single query and, hence, no error will occur.

For example, the following query is correct:

```
select id from remtab@rdb
where contains@rdb(text,'hello') > 0;
```

Related Topics

"Syntax for CONTEXT Index Type"

Oracle Text CONTAINS Query Operators

"The CONTEXT Grammar" topic in Oracle Text Application Developer's Guide

"SCORE"

1.5 CREATE INDEX

Use the CREATE INDEX statement to create an Oracle Text index.

This section describes the CREATE INDEX statement as it pertains to creating an Oracle Text domain index and composite domain index. See *Oracle Database SQL Language Reference* for a complete description of the CREATE INDEX statement.

Purpose

To create an Oracle Text index. An Oracle Text index is an Oracle Database domain index or composite domain index of type CONTEXT, CTXCAT, or CTXRULE. A domain index is an application-specific index. A **composite domain index** (CDI) is an Oracle Text index that not only indexes and processes a specified text column, but also indexes and processes FILTER BY and ORDER BY structured columns, which are specified during index creation.

Example

```
create table mytab
(item_id number,
  item_info varchar2(4000),
  item_supplier varchar2(250),
  item distributor varchar2(500));
```



create index idx on mytab(item_info) indextype is ctxsys.context
filter by item supplier order by item distributor;

You must create an appropriate Oracle Text index to enter CONTAINS, CATSEARCH, or MATCHES queries.

You cannot create an Oracle Text index on an index-organized table.

You can create the following types of Oracle Text indexes.

CONTEXT

A CONTEXT index is the basic type of Oracle Text index. This is an index on a text column. A CONTEXT index is useful when your source text consists of many large, coherent documents. Query this index with the CONTAINS operator in the WHERE clause of a SELECT statement. This index requires manual synchronization after DML. See Syntax for CONTEXT Index Type.

CTXCAT

The CTXCAT index is a combined index on a text column and one or more other columns. The CTXCAT type is typically used to index small documents or text fragments, such as item names, prices, and descriptions found in catalogs. Query this index with the CATSEARCH operator in the WHERE clause of a SELECT statement. This type of index is optimized for mixed queries. This index is transactional, automatically updating itself with DML to the base table. CTXCAT indexes are generally larger and slower to create and update than CONTEXT indexes, and have a narrower range of indexing options available. See Syntax for CTXCAT Index Type.

Note:

The Oracle Text indextype CTXCAT is deprecated with Oracle Database 23ai. The indextype itself, and it's operator CTXCAT, can be removed in a future release. Both CTXCAT and the use of CTXCAT grammar as an alternative grammar for CONTEXT queries is deprecated. Instead, Oracle recommends that you use the CONTEXT indextype, which can provide all the same functionality, except that it is not transactional. Near-transactional behavior in CONTEXT can be achieved by using SYNC (ON COMMIT) or, preferably, SYNC (EVERY [time-period]) with a short time period.

CTXCAT was introduced when indexes were typically a few megabytes in size. Modern, large indexes, can be difficult to manage with CTXCAT. The addition of index sets to CTXCAT can be achieved more effectively by the use of FILTER BY and ORDER BY columns, or SDATA, or both, in the CONTEXT indextype. CTXCAT is therefore rarely an appropriate choice. Oracle recommends that you choose the more efficient CONTEXT indextype.

CTXRULE

A CTXRULE index is used to build a document classification application. The CTXRULE index is an index created on a table of queries or a column containing a set of queries, where the queries serve as rules to define the classification criteria. Query this index with the MATCHES operator in the WHERE clause of a SELECT statement. See Syntax for CTXRULE Index Type.

Required Privileges



You do not need the CTXAPP role to create an Oracle Text index. If you have Oracle Database privileges to create an index on the text column, you have sufficient privilege to create a text index. The issuing owner, table owner, and index owner can all be different users, which is consistent with Oracle standards for creating regular indexes.



Whenever you create an Oracle Text index, a number of additional internal objects are created which have names prefixed with DR\$. These internal object names usually contain the index name. In some cases, the index name is shortened to fit in the object name. In such cases, the index ID is present in the object name to avoid naming conflicts with objects of other indexes.

Syntax for CONTEXT Index Type

Use a CONTEXT index to create an index on a text column. Query this index with the CONTAINS operator in the where clause of a Select statement. This index requires manual synchronization after DML.

```
CREATE INDEX [schema.]index ON [schema.]table(txt_column)
INDEXTYPE IS CTXSYS.CONTEXT [ONLINE]
[FILTER BY filter_column[, filter_column]...]
[ORDER BY oby_column[desc|asc][, oby_column[desc|asc]]...]
[LOCAL [PARTITION [partition] [PARAMETERS('paramstring')]]
[, PARTITION [partition] [PARAMETERS('paramstring')]])
[PARAMETERS(paramstring)] [PARALLEL n] [UNUSABLE]];
```

[schema.]index

Specifies the name of the Text index to create.

[schema.]table(txt_column)

Specifies the name of the table and column to index. txt_column is the name of the domain index column on which the CONTAINS() operator will be invoked.

Your table can optionally contain a primary key if you prefer to identify your rows as such when you use procedures in CTX_DOC. When your table has no primary key, document services identifies your documents by ROWID.



Primary keys of the following type are supported: NUMBER, VARCHAR2, DATE, CHAR, VARCHAR, and RAW.

The column that you specify must be one of the following types: CHAR, VARCHAR, VARCHAR2, BLOB, CLOB (limited to 4294967295 bytes), BFILE, XMLType, or URIType.



Note:

Starting with Oracle Database 12c Release 2 (12.2), an Oracle Text index cannot be created on a column with a declared collation other than <code>BINARY</code>, <code>USING_NLS_COMP</code>, <code>USING_NLS_SORT</code> or <code>USING_NLS_SORT_CS</code>. For all the supported collations, the Oracle Text behavior is the same.

The table that you specify can be a partitioned table. If you do not specify the LOCAL clause, then a global, nonpartitioned index is created.

The DATE, NUMBER, and nested table columns cannot be indexed. Object columns also cannot be indexed, but their attributes can be indexed, provided that they are atomic data types. Attempting to create an index on a Virtual Private Database (VPD) protected table will fail unless one of the following criteria is true:

- The VPD policy is created such that it does not apply to the INDEX statement type.
- The policy function returns a NULL predicate for the current user.
- The user (or index owner) is SYS.
- The user has the EXEMPT ACCESS POLICY privilege.

Note:

If you create VPD policies or use <code>DBMS_REDACT</code> after you create a context index on the base table, then the <code>DR\$</code> index tables like <code>\$I</code> will still contain the redacted column's indexed information. The <code>CONTAINS</code> queries also return results accordingly. To prevent indexing of sensitive data, either create the security redaction and VPD policies before creating a context index or rebuild the context index whenever security policies are added.

Indexes on multiple columns are not supported with CONTEXT index type. You must specify only one column in the column list.

Note:

With the CTXCAT index type, you can create indexes on text and structured columns. See "Syntax for CTXCAT Index Type"

Note:

Because a Transparent Data Encryption-enabled column does not support domain indexes, it cannot be used with Oracle Text. However, you can create an Oracle Text index on a column in a table stored in a Transparent Data Encryption-enabled tablespace.



ONLINE

Creates the index while enabling DML insertions/updates/deletions on the base table. During indexing, Oracle Text enqueues DML requests in a pending queue. At the end of the index creation, Oracle Text locks the base table. During this time, DML is blocked. You must synchronize the index in order for DML changes to be available.

Limitations

The following limitations apply to using ONLINE:

- At the very beginning or very end of the ONLINE process, DML might fail.
- ONLINE is supported for CONTEXT index only.

FILTER BY filter column

This is the structured indexed column on which a range or equality predicate in the WHERE clause of a mixed query will operate. You can specify one or more structured columns for filter_column, on which the relational predicates are expected to be specified along with the CONTAINS() predicate in a query.

The Cost-based Optimizer (CBO) will consider pushing down the structured predicates on these FILTER BY columns with the following relational operators: <, <=, =, >=, >, between, and LIKE (for VARCHAR2).

These columns can only be of CHAR, NUMBER, DATE, VARCHAR2, or RAW type. Additionally, CHAR, VARCHAR2 and VARCHAR2 types are supported only if the maximum length is specified and does not exceed 249 bytes. If the maximum length of a CHAR or VARCHAR2 column is specified in characters, for example, VARCHAR2 (50 CHAR), then it cannot exceed FLOOR (249/ max_char_width), where max_char_width is the maximum width of any character in the database character set. For example, the maximum specified column length cannot exceed 62 characters, if the database character set is AL32UTF8. The ADT attributes of supported types (CHAR, NUMBER, DATE, VARCHAR2, or RAW) are also allowed. An error is raised for all other data types. Expressions, for example, func (cola), and virtual columns are not allowed. txt column is allowed in the FILTER BY column list.

DML operations on FILTER BY columns are always transactional.

ORDER BY oby column

This is the structured indexed column on which a structured ORDER BY mixed query will be based. A list of structured *oby_columns* can be specified in the ORDER BY clause of a CONTAINS() query.

These columns can only be of CHAR, NUMBER, DATE, VARCHAR2, or RAW type. VARCHAR2 and RAW columns longer than 249 bytes are truncated to the first 249 bytes. Expressions, for example, func (cola), and virtual columns are not allowed.

The order of the specified columns matters. The Cost-based Optimizer (CBO) will consider pushing the sort into the composite domain index only if the ORDER BY clause in the text query contains:

- Entire ordered ORDER BY columns declared by the ORDER BY clause during the CREATE INDEX statement
- Only the prefix of the ordered ORDER BY columns declared by the ORDER BY clause during the CREATE INDEX statement
- The score followed by the prefix of the ordered ORDER BY columns declared by the ORDER BY clause during the CREATE INDEX statement
- The score following the prefix of the ordered ORDER BY columns declared by the ORDER BY clause during the CREATE INDEX statement



The following example illustrates Cost-based Optimizer (CBO) behavior with regard to ORDER BY columns:

```
CREATE INDEX foox ON foo(D) INDEXTYPE IS CTXSYS.CONTEXT FILTER BY B, C ORDER BY A, B desc;
```

Consider the following query:

```
SELECT A, SCORE(1) FROM foo WHERE CONTAINS(D, 'oracle',1)>0 AND C>100 ORDER BY col list;
```

Note:

If you set NLS_SORT or NLS_COMP parameters (that is, alter session set NLS_SORT = <some lang>;), then CBO will not push the sort or related structured predicate into the CDI. This behavior is consistent with regular optimized for search SDATA indexes.

The Cost-based Optimizer (CBO) will consider pushing the sort into the composite domain index (CDI) if col list has the following values:

```
A
A,B
SCORE(1), A
SCORE(1), A, B
A, SCORE(1)
A, B, SCORE(1)
```

The CBO will not consider to push the sort into the CDI if col list has the following values:

```
B
B,A
SCORE(1), B
B, SCORE(1)
A, B, C
A, B asc
```

(or simply A, B)

Expressions, for example, func(cola), are not allowed. txt_column appearing in the ORDER BY column list is allowed. DML operations on ORDER BY columns are always transactional.

Limitations

The following limitations apply to FILTER BY and ORDER BY:

- A structured column is allowed in FILTER BY and ORDER BY clauses. However, a column that is mapped to MDATA in a FILTER BY clause cannot also appear in the ORDER BY clause. An error will be raised in this case.
- The maximum length for CHAR, VARCHAR2, and RAW columns cannot be greater than 249 for FILTER BY columns. For ORDER BY columns, the data is truncated at 249 characters.

The total number of CDI (FILTER BY and ORDER BY) is 32.

Note:

In a CDI, if the indexed column is also a FILTER BY or ORDER BY column, then when you update the main indexed column, the updates to the FILTER BY or ORDER BY columns are not transactional.

Note:

- As with concatenated optimized for search SDATA indexes or bitmap indexes, performance degradation may occur in DML as the number of FILTER BY and ORDER BY columns increases.
- Mapping a FILTER BY column to MDATA is not recommended if the FILTER BY
 column contains sequential values or has very high cardinality. Doing so can
 result in a very long and narrow \$I table and reduced \$X performance. An
 example is a column of type DATE. For columns of this type, mapping to SDATA is
 recommended.

Note:

An index table with the name DR\$indextable\$S is created to store FILTER BY and ORDER BY columns that are mapped to SDATA sections. If nothing is mapped to an SDATA section, then the \$S table will not be created.

\$S table contains the following columns:

- SDATA ID number is the internal SDATA section ID.
- SDATA LAST number, the last document ID, which is analogous to token last.
- SDATA_DATA RAW(2000), the compressed SDATA values. Note that if \$s is created on a tablespace with 4K database block size, then it will be defined as RAW(1500).

Restriction: For performance reasons, \$s table must be created on a tablespace with db block size >= 4K without overflow segment and without PCTTHRESHOLD clause. If \$s is created on a tablespace with db block size < 4K, or is created with an overflow segment or with a PCTTHRESHOLD clause, then appropriate errors will be raised during the CREATE INDEX statement.

Restrictions on exporting and importing text tables with composite domain index created with Filter by and/or order by clauses are as follows:

Oracle recommends that you use Oracle Data Pump Import (impdp) and Oracle Data
 Pump Export (expdp) utilities for importing and exporting Oracle Text indexes.



- To export a text table with composite domain index, you must use Data Pump Export and Import utilities (invoked with the expdp and impdp commands, respectively) or DBMS DATAPUMP PL/SQL package.
- The original Oracle Database Export (exp) utility is desupported in Oracle Database 23ai.

See Also:

ADD_SDATA_COLUMN in CTX_DDL Package

Limitations of using ALTER INDEX and ALTER TABLE with FILTER BY and ORDER BY columns of the composite domain index, which are imposed by Extensible Indexing Framework in Oracle Database:

(These limitations are imposed by Extensible Indexing Framework in Oracle Database.)

- Using ALTER INDEX to add or drop FILTER BY and ORDER BY columns is currently not supported. You must re-create the index to add or drop FILTER BY or ORDER BY columns.
- To use ALTER TABLE MODIFY COLUMN to modify the datatype of a column that has the
 composite domain index built on it, you must first drop the composite domain index before
 modifying the column.
- To use ALTER TABLE DROP COLUMN to drop a column that is part of the composite domain index, you must first drop the composite domain index before dropping the index column.

The following limitations apply to FILTER BY and ORDER BY when used with PL/SQL packages:

- Mapping FILTER BY columns to sections is optional. If section mapping does not exist for a FILTER BY column, then it is mapped to an SDATA section by default. The section name assumes the name of the FILTER BY column.
- If a section group is not specified during the CREATE INDEX clause of a composite domain index, then system default section group settings are used. An SDATA section is created for each of the FILTER BY and ORDER BY columns.

Note:

Because a section name does not allow certain special characters and is case-insensitive, if the column name is case-sensitive or contains special characters, then an error will be raised. To work around this problem, you must map the column to an MDATA or SDATA section before creating the index. See CTX DDL.ADD MDATA COLUMN or CTX DDL.ADD SDATA COLUMN.

- An error is raised if a column that is mapped to an MDATA section also appears in the ORDER BY column clause.
- Column section names are unique to their section group. That is, you cannot have an MDATA column section named FOO if you already have an MDATA column section named FOO. Nor can you have a field section named FOO if you already have an SDATA column section named FOO. This is true whether it is implicitly created (by CREATE INDEX for FILTER BY or ORDER BY clauses) or explicitly created (by CTX DDL.ADD SDATA COLUMN).



- One section name can be mapped to only one FILTER BY column, and vice versa. Mapping
 a section to more than one column, or mapping a column to more than one section is not
 allowed.
- Column sections can be added to any type of section group, including the NULL section group.
- If a section group with sections added by the CTX_DDL.ADD_MDATA_COLUMN or CTX_DDL.ADD_SDATA_COLUMN packages is specified for a CREATE INDEX statement without a FILTER BY clause, then the mapped column sections will be ignored. However, the index will still get created without those column sections. The same is true for a FILTER BY clause that does not contain mapped columns in the specified section group.

✓ See Also:
CTX DDL.ADD SDATA COLUMN

LOCAL [PARTITION [partition] [PARAMETERS('paramstring')]

Specifies a local partitioned context index on a partitioned table. The partitioned table must be partitioned by range. Hash, composite, and list partitions are not supported.

You can specify the list of index partition names with *partition_name*. If you do not specify a partition name, then the system assigns one. The order of the index partition list must correspond to the table partition order.

The PARAMETERS clause associated with each partition specifies the parameters string specific to that partition. You can only specify sync (manual|every |on commit), memory and storage for each index partition.

The PARAMETERS clause also supports the POPULATE and NOPOPULATE arguments. See POPULATE | NOPOPULATE.

Query the views CTX_INDEX_PARTITIONS or CTX_USER_INDEX_PARTITIONS to find out index partition information, such as index partition name, and index partition status.

See Also:

Creating a Local Partitioned Index

Query Performance Limitation with Partitioned Index

For optimal performance when querying a partitioned index with an ORDER BY SCORE clause, query the partition. If you query the entire table and use an ORDER BY SCORE clause, the query might not perform optimally unless you include a range predicate that can limit the query to the fewest number of partitions, which is optimally a single partition.

See Also:

Query Performance Limitation with a Partitioned Index

PARALLEL n

Optionally specifies the parallel degree for parallel indexing. The actual degree of parallelism might be smaller depending on your resources. You can use this parameter on nonpartitioned



tables. However, creating a nonpartitioned index in parallel does not turn on parallel query processing. Parallel indexing is supported for creating a local partitioned index. The indexing memory size specified in the parameter clause applies to each parallel worker. For example, if indexing memory size is specified in the parameter clause as 500M and parallel degree is specified as 2, then you must ensure that there is at least 1GB of memory available for indexing.

See Also:

- Parallel Indexing
- Creating a Local Partitioned Index in Parallel
- The "Performance Tuning" chapter in Oracle Text Application Developer's Guide

Performance

Parallel indexing can speed up indexing when you have large amounts of data to index and when your operating system supports multiple CPUs.

Note:

Using PARALLEL to create a local partitioned index that enables parallel queries. (Creating a nonpartitioned index in parallel does not turn on parallel query processing.)

Parallel querying degrades query throughput especially on heavily loaded systems. Because of this, Oracle recommends that you disable parallel querying after creating a local index. To do so, use the ALTER INDEX NOPARALLEL statement.

For more information on parallel querying, see the "Performance Tuning" chapter in *Oracle Text Application Developer's Guide*.

Limitations

Parallel indexing is supported only for the CONTEXT index type.

UNUSABLE

Creates an unusable index. This creates index metadata only and exits immediately. You might create an unusable index when you need to create a local partitioned index in parallel.



"Creating a Local Partitioned Index in Parallel"

PARAMETERS(paramstring)

Optionally specify indexing parameters in paramstring. You can specify preferences owned by another user using the user.preference notation.



The syntax for paramstring is as follows:

```
paramstring =
'[ASYNCHRONOUS UPDATE | SYNCHRONOUS UPDATE]
 [DATASTORE datastore pref]
 [FILTER filter pref]
 [CHARSET COLUMN charset column name]
 [FORMAT COLUMN format column name]
 [SAVE COPY COLUMN save copy column name]
 [LEXER lexer pref]
 [LANGUAGE COLUMN language column name]
 [WORDLIST wordlist pref]
 [STORAGE storage pref]
 [STOPLIST stoplist]
 [SECTION GROUP section group]
 [MEMORY memsize]
 [POPULATE | NOPOPULATE]
 [MAINTENANCE AUTO | MAINTENANCE MANUAL]
 [SYNC (MANUAL | EVERY "interval-string" | ON COMMIT)]
 [TRANSACTIONAL]
 [OPTIMIZE (MANUAL | AUTO DAILY | EVERY "interval-string")]'
```

Create datastore, filter, lexer, wordlist, and storage preferences with CTX_DDL.CREATE_PREFERENCE and then specify them in the paramstring.

Note:

The combination of ASYNCHRONOUS_UPDATE and TRANSACTIONAL parameters is not supported for context indexes.

Note:

When you specify no paramstring, Oracle Text uses the system defaults. For more information about these defaults, see "Default Index Parameters".

ASYNCHRONOUS_UPDATE | SYNCHRONOUS_UPDATE

Specifies whether Oracle Text must retain old index entries for documents in which the indexed column was updated. The default is <code>SYNCHRONOUS_UPDATE</code> which indicates that index updates are synchronous and that old index entries are unavailable for search operations until the index is synchronized.

ASYNCHRONOUS_UPDATE indicates that until the index is synchronized, search queries will use the old index entries to return the old document content. After index synchronization, the rebuilt index is used to return the updated document content.

Asynchronous updates are not supported for DML operations that cause row movement. This option cannot be set at the partition level.

The following example creates a CONTEXT index idx for which asynchronous update is enabled.

CREATE INDEX myidx ON mytabl(item_info) INDEXTYPE IS CTXSYS.CONTEXT PARAMETERS('asynchronous update');



Note:

The ASYNCHRONOUS_UPDATE setting of the CONTEXT indextype is deprecated in Oracle Database 23ai, and can be ignored or removed in a future release.

Oracle can ignore or remove this attribute in a future release. Oracle recommends

that you allow this value to be set to its default value, SYNCHRONOUS_UPDATE. To avoid unexpected loss of results during updates, use SYNC (ON COMMIT) or SYNC (EVERY [time-period]) with a short time period.

The ASYNCHRONOUS_UPDATE setting was introduced as a workaround for the fact that updates are implemented as "delete followed by insert," and that deletes are immediate (on commit), while inserts are only performed during an index sync. However, this setting is incompatible with several other index options. Oracle recommends that you discontinue its use.

DATASTORE datastore pref

Specifies the name of your datastore preference. Use the datastore preference to specify where your text is stored. See "Datastore Types".

FILTER filter pref

Specifies the name of your filter preference. Use the filter preference to specify how to filter formatted documents to plain text or HTML. See "Filter Types".

CHARSET COLUMN charset column name

Specifies the name of the character set column. This column must be in the same table as the text column, and it must be of type CHAR, VARCHAR, or VARCHAR2. Use this column to specify the document character set for conversion to the database character set. The value is case-insensitive. You must specify a globalization support character set string, such as JA16EUC. When the document is plain text or HTML, the AUTO_FILTER and CHARSET filters use this column to convert the document character set to the database character set for indexing. Use this column when you have plain text or HTML documents with different character sets or in a character set different from the database character set.

Setting $\mbox{NLS_LENGTH_SEMANTICS}$ parameter to \mbox{CHAR} is not supported at the database level. This parameter is supported for the following columns:

• The CHARSET COLUMN, for example:

VARCHAR2 < size> CHAR CHAR < size> CHAR

- An index created on a VARCHAR2 and CHAR column
- VARCHAR2 and CHAR columns for FILTER BY and ORDER BY clauses of CREATE INDEX
- FORMAT COLUMN



Note:

- Documents are not marked for re-indexing when only the character set column changes. The indexed column must be updated to flag the re-index.
- The NLS_LENGTH_SEMANTICS = CHAR parameter is supported at the column level only, and is not supported at the database level, as described in this section.

FORMAT COLUMN format column name

Specifies the name of the format column. The format column must be in the same table as the text column and it must be CHAR, VARCHAR, or VARCHAR2 type.

FORMAT COLUMN determines how a document is filtered, or, in the case of the IGNORE value, if it is to be indexed.

AUTO_FILTER uses the format column when filtering documents. Use this column with heterogeneous document sets to optionally bypass filtering for plain text or HTML documents. In the format column, you can specify one of the following options:

- TEXT
- BINARY
- IGNORE

The TEXT option indicates that the document is either plain text or HTML. When TEXT is specified, the document is not filtered, but may have the character set converted. The BINARY option indicates that the document is a format supported by the AUTO_FILTER object other than plain text or HTML, for example PDF. BINARY is the default, if the format column entry cannot be mapped.

The IGNORE option indicates that the row is to be ignored during indexing. Use this value when you need to bypass rows that contain data incompatible with text indexing such as image data, or rows in languages that you do not want to process. The difference between documents with TEXT and IGNORE format column types is that the former are indexed but ignored by the filter, while the latter are not indexed at all. Thus, IGNORE can be used with any filter type.

Note:

Documents are not marked for re-indexing when only the format column changes. The indexed column must be updated to flag the re-index.

SAVE_COPY COLUMN save_copy_column_name

Specifies the name of the column that contains the preference of whether to save a copy of a document into the p index table during a search operation.

You can specify one of the following three options in the SAVE_COPY column: PLAINTEXT, FILTERED, or NONE.

The PLAINTEXT option indicates that the document should be stored as a plain text in the \$D index table. Specify this value when using the SNIPPET procedure.

The FILTERED option indicates that a filter preference should be applied on the text present in the document before storing it into the \$D index table. Specify this value when using the MARKUP procedure or the HIGHLIGHT procedure.

The NONE option indicates that a copy of the document should not be saved in the \$D index table. Specify this value for any of the following scenarios:



- when SNIPPET, MARKUP, or HIGHLIGHT procedure is not used.
- when the indexed column is either VARCHAR2 or CLOB.

LEXER lexer pref

Specifies the name of your lexer or multilexer preference. Use the lexer preference to identify the language of your text and how text is tokenized for indexing. See "Lexer Types".

LANGUAGE COLUMN language_column_name

Specifies the name of the language column when using a multi-lexer preference. See "MULTI LEXER".

This column must exist in the base table. It cannot be the same column as the indexed column. Only the first 30 bytes of the language column are examined for language identification.



Documents are not marked for re-indexing when only the language column changes. The indexed column must be updated to flag the re-index.

WORDLIST wordlist pref

Specifies the name of your wordlist preference. Use the wordlist preference to enable features such as fuzzy, stemming, and prefix indexing for better wildcard searching. See "Wordlist Type".

STORAGE storage_pref

Specifies the name of your storage preference for the Text index. Use the storage preference to specify how the index tables are stored. See "Storage Types".

STOPLIST stoplist

Specifies the name of your stoplist. Use stoplist to identify words that are not to be indexed. See $CTX_DDL.CREATE_STOPLIST$.

SECTION GROUP section_group

Specifies the name of your section group. Use section groups to create searchable sections in structured documents. See CTX_DDL.CREATE_SECTION_GROUP.

MEMORY memsize

Specifies the amount of run-time memory to use for indexing. The syntax for memsize is as follows:

memsize = number[K|M|G]

K stands for kilobytes, M stands for megabytes, and G stands for gigabytes.

The value you specify for memsize must be between 1M and the value of MAX_INDEX_MEMORY in the CTX_PARAMETERS view. To specify a memory size larger than the MAX_INDEX_MEMORY, you must reset this parameter with CTX_ADM.SET_PARAMETER to be larger than or equal to memsize.

The default is the value specified for Default Index memory in CTX Parameters.

The memsize parameter specifies the amount of memory Oracle Text uses for indexing before flushing the index to disk. Specifying a large amount memory improves indexing performance because there are fewer I/O operations and improves query performance and maintenance, because there is less fragmentation.



Specifying smaller amounts of memory increases disk I/O and index fragmentation, but might be useful when run-time memory is scarce.

POPULATE | NOPOPULATE

Specifies whether an index should be empty or populated. The default is POPULATE. The POPULATE and NOPOPULATE parameters are applicable to CONTEXT, CTXRULE, and SEARCH INDEX types.

Note:

 $\label{eq:populate} \begin{tabular}{ll} \verb|POPULATE| & \verb|POPULATE| & \verb|SET| & \verb|PARAMETER| & \verb|$

Empty indexes are populated by updates or inserts to the base table. You might create an empty index when you need to create your index incrementally or to selectively index documents in the base table. You might also create an empty index when you require only theme and Gist output from a document set.

Note that a populated index is created by default, unless you explicitly specify the NOPOPULATE keyword. The outputs of CTX_REPORT.CREATE_INDEX_SCRIPT and CTX_REPORT.DESCRIBE_INDEX include the NOPOPULATE keyword for such indexes.

MAINTENANCE AUTO | MAINTENANCE MANUAL

Specifies the maintenance type for synchronization of the CONTEXT index when there are inserts, updates, or deletes to the base table. The maintenance type specified for an index applies to all index partitions.

You can set one of the following maintenance types:

Maintenance Type	Description
MAINTENANCE AUTO	This is the default method for synchronizing Oracle Text CONTEXT and search indexes.
	This method sets your index to automatic maintenance, that is, the index is automatically synchronized in the background at optimal intervals.
	You do not need to manually configure a SYNC type or set any synchronization interval. The background mechanism automatically determines the synchronization interval and schedules background SYNC.INDEX operations by tracking the DML queue.
	Note : Review a list of requirements and restrictions for indexes in an automatic maintenance mode, as listed in <i>Oracle Text Application Developer's Guide</i> .
MAINTENANCE MANUAL	This method sets your index to manual maintenance. This is a non-automatic maintenance (synchronization) mode in which you can specify SYNC types, such as MANUAL, EVERY, or ON COMMIT.

SYNC (MANUAL | EVERY "interval-string" | ON COMMIT)

Specifies the SYNC type for synchronization of the CONTEXT index when there are inserts, updates, or deletes to the base table.

These SYNC settings are applicable only to the indexes that are set to manual maintenance.





By default, the CONTEXT and search indexes run in an automatic maintenance mode (MAINTENANCE AUTO), which means that your DMLs are automatically synchronized into the index in the background at optimal intervals. Therefore, you do not need to manually configure a SYNC method. However, if required, you can do so if you want to modify the default settings for an index.

You can specify one of the following SYNC methods:

SYNC Type	Description
MANUAL	This is the default synchronization method for CONTEXT index. In this method, automatic synchronization is not provided. You must manually synchronize the index with CTX_DDL.SYNC_INDEX.
EVERY "interval-string"	The default synchronization interval is set to 30 seconds. Automatically synchronizes the index at a regular interval specified by the value of <i>interval-string</i> , which takes the same syntax as that for scheduler jobs. Automatic synchronization using EVERY requires that the index creator have CREATE JOB privileges. Ensure that <i>interval-string</i> is set to a considerable time period that any previous sync jobs will have completed; otherwise, the sync job might stop responding. <i>interval-string</i> must be enclosed in double quotes, and any single quote within <i>interval-string</i> must be preceded by the escape character with another single quote. See Enabling Automatic Index Synchronization at Regular Intervals for an example of automatic sync syntax.
ON COMMIT	Synchronizes the index immediately after a commit transaction. The commit transaction does not return until the sync is complete. Before Oracle Database Release 18c, the synchronization was performed as a separate transaction. There was a time period, usually small, when the data was committed but index changes were not. Starting with Oracle Database Release 18c, the synchronization is performed as part of the same transaction. The operation uses the memory specified with the <i>memory</i> parameter. Before Oracle Database Release 18c, the sync operation had its own transaction context. If the operation failed, the data transaction still committed. Starting with Oracle Database Release 18c, if there is an irrecoverable index synchronization error, the entire data transaction is rolled back. Recoverable (individual row) synchronization errors are logged in the CTX_USER_INDEX_ERRORS view but the transaction still completes. See Viewing Index Errors. See Enabling Automatic Index Synchronization at Regular Intervals for an example of ON COMMIT syntax.



Each partition of a locally partitioned index can have its own type of sync (ON COMMIT, EVERY, or MANUAL). The type of sync specified in primary parameter strings applies to all index partitions unless a partition specifies its own type.

With automatic (EVERY) synchronization, users can specify memory size and parallel synchronization. That syntax is:

```
... EVERY interval string MEMORY mem size PARALLEL paradegree ...
```

The ON COMMIT synchronizations can be run only serially and must use the same memory size that was specified at index creation.

See Also:

- Oracle Database Administrator's Guide for information about job scheduling
- Oracle Database PL/SQL Packages and Types Reference for information about DBMS_SCHEDULER

TRANSACTIONAL

Specifies that documents can be searched immediately after they are inserted or updated. If a text index is created with TRANSACTIONAL enabled, then, in addition to processing the synchronized rowids already in the index, the CONTAINS operator will process unsynchronized rowids as well. Oracle Text does in-memory indexing of unsynchronized rowids and processes the query against the in-memory index.

TRANSACTIONAL is an index-level parameter and does not apply at the partition level. You must still synchronize your text indexes from time to time (with CTX_DDL.SYNC_INDEX) to bring pending rowids into the index. Query performance degrades as the number of unsynchronized rowids increases. For that reason, Oracle recommends setting up your index to use automatic synchronization with the EVERY or ON COMMIT parameter. (See "SYNC (MANUAL | EVERY "interval-string" | ON COMMIT)".)

Transactional querying for indexes that have been created with the TRANSACTIONAL parameter can be turned on and off (for the duration of a user session) with the PL/SQL variable CTX_QUERY.disable_transactional_query. This is useful, for example, if you find that querying is slow due to the presence of too many pending rowids. Here is an example of setting this session variable:

```
exec ctx query.disable transactional query := TRUE;
```

If the index uses ${\tt AUTO_FILTER}$, queries involving unsynchronized rowids will require filtering of unsynchronized documents.

OPTIMIZE (MANUAL | AUTO_DAILY | EVERY "interval-string")

Specify OPTIMIZE to enable automatic background index optimization. You can specify any one of the following OPTIMIZE methods:

OPTIMIZE Type	Description
MANUAL	Provides no automatic optimization. You must manually
	optimize the index with CTX_DDL.OPTIMIZE_INDEX.



OPTIMIZE Type	Description
AUTO_DAILY	 When you specify OPTIMIZE (AUTO_DAILY) in the create index parameter list, a repeatedly running optimize token job and a repeatedly running optimize full job are scheduled for each index and partition: The Optimize token job is scheduled to run weekly from 12 A.M. every Saturday night to optimize \$S* tables. This job runs on tables with non-JSON data type (VARCHAR2, CLOB, or BLOB) to optimize the top 10 most fragmented tokens (determined automatically). The Optimize full job is scheduled to run every midnight from 12 A.M. to 3 A.M. except on Saturday night. Jobs that are not started before 3 A.M. are skipped. These skipped jobs are started before the other jobs that are scheduled to run at 12 A.M. the next day. This job runs on tables with JSON data type or the IS JSON check constraint. Existing indexes do not have OPTIMIZE (AUTO_DAILY) by default. You must use ALTER INDEX to enable automatic background index optimization.
EVERY "interval-string"	Automatically runs at a regular interval specified by the value <i>interval-string</i> , which takes the same syntax as scheduler jobs. • The Optimize token job is scheduled for tables with non-JSON data type. This job runs optimize token for the top 10 most fragmented tokens at an interval specified by the user. • The Optimize full job is scheduled for tables with JSON data type or the IS JSON check constraint. This job runs optimize full weekly at 12 A.M. every Saturday night for \$S* tables. Ensure that <i>interval-string</i> is set to a considerable time period so that any previous optimize jobs are complete. The <i>interval-string</i> value must be enclosed in double quotes, and any single quote within <i>interval-string</i> must be preceded by the escape character with another single quote. If multiple indexes use the OPTIMIZE EVERY "interval-string" option, then different jobs are created for each index. These jobs are run concurrently.

With ${\tt AUTO_DAILY} \mid {\tt EVERY} \; "interval-string"$ setting, you can specify parallel optimization. That syntax is:

... [AUTO DAILY | EVERY "interval-string"] PARALLEL paradegree ...

CREATE INDEX: CONTEXT Index Examples

The following sections give examples of creating a ${\tt CONTEXT}$ index.

Creating CONTEXT Index Using Default Preferences

The following example creates a CONTEXT index called myindex on the docs column in mytable. Default preferences are used.

CREATE INDEX myindex ON mytable (docs) INDEXTYPE IS ctxsys.context;

See Also:

- Oracle Text Application Developer's Guide
- For more information about default settings, see "Default Index Parameters"

Creating CONTEXT Index with Custom Preferences

The following example creates a CONTEXT index called myindex on the docs column in mytable. The index is created with a custom lexer preference called my_lexer and a custom stoplist called my_stop.

This example also assumes that the preference and stoplist were previously created with CTX_DDL.CREATE_PREFERENCE for my_lexer, and CTX_DDL.CREATE_STOPLIST for my_stop. Default preferences are used for the unspecified preferences.

```
CREATE INDEX myindex ON mytable(docs) INDEXTYPE IS ctxsys.context PARAMETERS('LEXER my lexer STOPLIST my stop');
```

Any user can use any preference. To specify preferences that exist in another user's schema, add the user name to the preference name. The following example assumes that the preferences my lexer and my stop exist in the schema that belongs to user kenny:

```
CREATE INDEX myindex ON mytable(docs) INDEXTYPE IS ctxsys.context PARAMETERS('LEXER kenny.my lexer STOPLIST kenny.my stop');
```

Enabling Automatic Index Synchronization at Regular Intervals

You can create your index and specify that the index be synchronized at regular intervals for insertions, updates and deletions to the base table. To do so, create the index with the SYNC (EVERY "interval-string") parameter.

To use job scheduling, you must log in as a user who has DBA privileges and then grant CREATE JOB privileges.

The following example creates an index and schedules three synchronization jobs for three index partitions. The first partition uses ON COMMIT synchronization. The other two partitions are synchronized by jobs that are scheduled to be executed every Monday at 3 P.M.

```
CONNECT system/password

GRANT CREATE JOB TO dr_test

CREATE INDEX tdrmauto02x ON tdrmauto02(text)
   INDEXTYPE IS CTXSYS.CONTEXT local
   (PARTITION tdrm02x_i1 PARAMETERS('
   MEMORY 20m SYNC(ON COMMIT)'),
   PARTITION tdrm02x i2,
```



```
PARTITION tdrm02x_i3) PARAMETERS('
SYNC (EVERY "NEXT_DAY(TRUNC(SYSDATE), ''MONDAY'') + 15/24")
');
```

See Oracle Database Administrator's Guide for information about job scheduling syntax.

Enabling Automatic Background Index Optimization

The following example creates an index and schedules a repeatedly running optimize token job at 12 A.M. every midnight and a repeatedly running optimize full job running at 12 A.M. every Saturday night.

```
CREATE TABLE mytable (
    text VARCHAR2(30)
);

CREATE INDEX myindex ON mytable(text)
    INDEXTYPE IS CTXSYS.CONTEXT
    PARAMETERS('OPTIMIZE (EVERY "FREQ=DAILY; BYHOUR=0")');
```

Creating CONTEXT Index with Multilexer Preference

The multilexer preference decides which lexer to use for each row based on a language column. This is a character column in the table that stores the language of the document in the text column. For example, create the table <code>globaldoc</code> to hold documents of different languages:

```
CREATE TABLE globaldoc (
   doc_id NUMBER PRIMARY KEY,
   lang VARCHAR2(10),
   text CLOB
);
```

Assume that <code>global_lexer</code> is a multilexer preference you created. To index the <code>global_doc</code> table, specify the multilexer preference and the name of the language column as follows:

```
CREATE INDEX globalx ON globaldoc(text) INDEXTYPE IS ctxsys.context PARAMETERS ('LEXER global lexer LANGUAGE COLUMN lang');
```

```
See Also:
```

"MULTI LEXER" for more information about creating multilexer preferences

Creating a Local Partitioned Index

The following example creates a text table that is partitioned into three, populates it, and then creates a partitioned index:

```
PROMPT create partitioned table and populate it

CREATE TABLE part tab (a int, b varchar2(40)) PARTITION BY RANGE(a)
```



```
(partition p_tab1 values less than (10),
  partition p_tab2 values less than (20),
  partition p_tab3 values less than (30));

PROMPT create partitioned index
CREATE INDEX part_idx on part_tab(b) INDEXTYPE IS CTXSYS.CONTEXT
  LOCAL (partition p_idx1, partition p_idx2, partition p_idx3);
CREATE INDEX part idx on part tab(b) INDEXTYPE IS CTXSYS.CONTEXT LOCAL;
```

Perform either of the following actions if there is going to be more than 10000 partitions:

- If you need to create a CONTEXT index with more than 10000 partitions, then you must use event 30579, level 2147483648 during index creation.
- If an index is already created and it has more than 10000 partitions, then you must recreate the index after running the following command:

```
alter SYSTEM set events '30579 trace name context forever, level 2147483648';
```



MOS note 2671924.1

Note:

The limit for the number of partitions in Oracle Text is the same as the maximum number of partitions per table in Oracle Database.

Using FILTER BY and ORDER BY Clauses

The following example creates an index on table *docs* and orders the documents by author's publishing date.

First, create the table:

```
CREATE TABLE docs (
docid NUMBER,
pub_date DATE,
author VARCHAR2(30),
category VARCHAR2(30),
document CLOB
);
```

Create the index with FILTER BY and ORDER BY clauses:

```
CREATE INDEX doc_idx on docs(document) indextype is ctxsys.context
FILTER BY category, author
ORDER BY pub_date desc, docid
PARAMETERS ('memory 500M');
```



Parallel Indexing

Parallel indexing can improve index performance when you have multiple CPUs.

To create an index in parallel, use the PARALLEL clause with a parallel degree. This example uses a parallel degree of 3:

```
CREATE INDEX myindex ON mytab(pk) INDEXTYPE IS ctxsys.context PARALLEL 3;
```

Creating a Local Partitioned Index in Parallel

Creating a local partitioned index in parallel can improve performance when you have multiple CPUs. With partitioned tables, you can divide the work. You can create a local partitioned index in parallel in two ways:

- Use the PARALLEL clause with the LOCAL clause in the CREATE INDEX statement. In this
 case, the maximum parallel degree is limited to the number of partitions you have. See
 "Parallelism with CREATE INDEX".
- Create an unusable index first, then run the DBMS_PCLXUTIL.BUILD_PART_INDEX utility. This method can result in a higher degree of parallelism, especially if you have more CPUs than partitions. See "Parallelism with DBMS_PCLUTIL.BUILD_PART_INDEX".

If you attempt to create a local partitioned index in parallel, and the attempt fails, you may see the following error message:

```
ORA-29953: error in the execution of the ODCIIndexCreate routine for one or more of the index partitions
```

To determine the specific reason why the index creation failed, query the CTX USER INDEX ERRORS view.

Parallelism with CREATE INDEX

You can achieve local index parallelism by using the PARALLEL and LOCAL clauses in the CREATE INDEX statement. In this case, the maximum parallel degree is limited to the number of partitions that you have.

The following example creates a table with three partitions, populates them, and then creates the local indexes in parallel with a degree of 2:

```
create table part_tab3(id number primary key, text varchar2(100))
partition by range(id)
(partition p1 values less than (1000),
  partition p2 values less than (2000),
  partition p3 values less than (3000));

begin
  for i in 0..2999
  loop
     insert into part_tab3 values (i,'oracle');
  end loop;
end;
/
create index part_tab3x on part_tab3(text)
indextype is ctxsys.context local (partition part tabx1,
```



```
partition part_tabx2,
partition part_tabx3)
```

Parallelism with DBMS_PCLUTIL.BUILD_PART_INDEX

You can achieve local index parallelism by first creating an unusable CONTEXT index, and then running the DBMS_PCLUTIL.BUILD_PART_INDEX utility. This method can result in a higher degree of parallelism, especially when you have more CPUs than partitions.

In this example, the base table has three partitions. We create a local partitioned unusable index first, then run <code>DBMS_PCLUTIL.BUILD_PART_INDEX</code>, which builds the 3 partitions in parallel (referred to as inter-partition parallelism). Also, inside each partition, index creation proceeds in parallel (called intra-partition parallelism) with a parallel degree of 2. Therefore, the total parallel degree is 6 (3 times 2).

```
create table part tab3(id number primary key, text varchar2(100))
partition by range (id)
(partition p1 values less than (1000),
partition p2 values less than (2000),
 partition p3 values less than (3000));
begin
  for i in 0..2999
  loop
      insert into part tab3 values (i,'oracle');
  end loop;
end;
create index part tab3x on part tab3(text)
indextype is ctxsys.context local (partition part tabx1,
                                   partition part tabx2,
                                   partition part tabx3)
unusable;
exec dbms pclxutil.build part index(jobs per batch=>3,
  procs per job=>2,
  tab name=>'PART TAB3',
  idx_name=>'PART TAB3X',
  force opt=>TRUE);
```

Viewing Index Errors

After a CREATE INDEX or ALTER INDEX operation, you can view index errors with Oracle Text views. To view errors on your indexes, query the CTX_USER_INDEX_ERRORS view. To view errors on all indexes as CTXSYS, query the CTX_INDEX_ERRORS view.

For example, to view the most recent errors on your indexes, enter the following statement:

```
SELECT err_timestamp, err_text FROM ctx_user_index_errors ORDER BY err timestamp DESC;
```

Deleting Index Errors



To clear the index error view, enter the following statement:

```
DELETE FROM ctx user index errors;
```

Syntax for CTXCAT Index Type

Combines an index on a text column and one or more other columns. Query this index with the CATSEARCH operator in the WHERE clause of a SELECT statement. This type of index is optimized for mixed queries. This index is transactional, automatically updating itself with DML to the base table.

```
CREATE INDEX [schema.]index on [schema.]table(column) INDEXTYPE IS ctxsys.ctxcat
[PARAMETERS('[index set index_set]
[lexer lexer_pref]
[storage storage_pref]
[stoplist stoplist]
[section group sectiongroup_pref]
[wordlist wordlist_pref]
[memory memsize]');
```

[schema.]table(column)

Specifies the name of the table and column to index.

The column that you specify when you create a CTXCAT index must be of type CHAR or VARCHAR2. No other types are supported for CTXCAT.

Attempting to create an index on a Virtual Private Database (VPD) protected table will fail unless one of the following options is true:

- The VPD policy is created such that it does not apply to INDEX statement type, which is the default
- The policy function returns a null predicate for the current user.
- The user (index owner) is SYS.
- The user has the EXEMPT ACCESS POLICY privilege.

Supported CTXCAT Preferences

index set index set

Specifies the index set preference to create the CTXCAT index. Index set preferences name the columns that make up your subindexes. Any column that is named in an index set column list cannot have a NULL value in any row of the base table, or else you get an error.

Always ensure that your columns have non-null values before and after indexing. See "Creating a CTXCAT Index".

Index Performance and Size Considerations

Although a CTXCAT index offers query performance benefits, creating this type of index has its costs. The time that it takes Oracle Text to create a CTXCAT index depends on the total size of the index.

The total size of a CTXCAT index is directly related to:

- Total text to be indexed
- Number of component indexes in the index set



Number of columns in the base table that make up the component indexes

Having many component indexes in your index set also degrades DML performance because more indexes must be updated.

Because of these added costs in creating a CTXCAT index, you should carefully consider the query performance benefit that each component index gives your application before adding it to your index set.



Oracle Text Application Developer's Guide for more information about creating CTXCAT indexes and the benefits

Other CTXCAT Preferences

When you create an index of type CTXCAT, you can use the supported index preferences listed in Table 1-12 in the parameters string.

Table 1-12 Supported CTXCAT Index Preferences

Preference Class	Supported Types
Datastore	This preference class is not supported for CTXCAT.
Filter	This preference class is not supported for CTXCAT.
Lexer	BASIC_LEXER (index_themes attribute not supported)
	CHINESE_LEXER
	CHINESE_VGRAM_LEXER
	JAPANESE_LEXER
	JAPANESE_VGRAM_LEXER
	KOREAN_MORPH_LEXER
Wordlist	BASIC_WORDLIST
Storage	BASIC_STORAGE
Stoplist	Supports single language stoplists only (BASIC_STOPLIST type).
Section Group	Only Field Section is supported for CTXCAT.

Unsupported Preferences and Parameters

When you create a CTXCAT index, you cannot specify datastore and filter preferences. For section group preferences, only the field section preference is supported. You also cannot specify language, format, or charset columns as with a CONTEXT index.

Creating a CTXCAT Index

This section gives a brief example for creating a CTXCAT index. For a more complete example, see *Oracle Text Application Developer's Guide*.

Consider a table called AUCTION with the following schema:

```
create table auction(item_id number,
title varchar2(100),
category id number,
```



```
price number,
bid close date);
```

Assume that queries on the table involve a mandatory text query clause and optional structured conditions on price. Results must be sorted based on bid_close. This means that an index to support good response time for the structured and sorting criteria is required.

You can create a catalog index to support the different types of structured queries a user might enter. For structured queries, a CTXCAT index improves query performance over a context index.

To create the indexes, first, create the index set preference, next, optionally, add the storage preference, and, finally, add the required indexes to it:

```
begin
ctx_ddl.create_index_set('auction_iset');
ctx_ddl.add_index('auction_iset','bid_close');
ctx_ddl.add_index('auction_iset','price, bid_close');
end;
```

Optionally, create the storage preference:

Then, create the CTXCAT index with the CREATE INDEX statement as follows:

```
create index auction_titlex on AUCTION(title) indextype is CTXSYS.CTXCAT
parameters ('index set auction_iset storage auction_st_pref');
```

Querying a CTXCAT Index

To query the title column for the word *pokemon*, enter regular and mixed queries as follows:

```
select * from AUCTION where CATSEARCH(title, 'pokemon',NULL)> 0;
select * from AUCTION where CATSEARCH(title, 'pokemon', 'price < 50 order by
bid close desc')> 0;
```

See Also:

Oracle Text Application Developer's Guide for a complete CTXCAT example

Syntax for CTXRULE Index Type

The CTXRULE type is an index on a column containing a set of queries. Query this index with the MATCHES operator in the WHERE clause of a SELECT statement.

```
CREATE INDEX [schema.]index on [schema.]table(rule_col) INDEXTYPE IS ctxsys.ctxrule

[PARAMETERS ('[lexer lexer_pref] [storage storage_pref] [section group section_pref] [wordlist wordlist_pref] [classifier classifier_pref]');

[PARALLEL n];
```

[schema.]table(column)

Specifies the name of the table and rule column to index. The rules can be query compatible strings, query template strings, or binary Support Vector Machine rules.

The column you specify when you create a CTXRULE index must be VARCHAR2, CLOB or BLOB. No other types are supported for the CTXRULE type.

Attempting to create an index on a Virtual Private Database (VPD) protected table will fail unless one of the following is true:

- The VPD policy does not have the INDEX statement type turned on (which is the default).
- The policy function returns a null predicate for the current user.
- The user (index owner) is SYS.
- The user has the EXEMPT ACCESS POLICY privilege.

lexer pref

Specifies the lexer preference to be used for processing queries and later for the documents to be classified with the MATCHES function.

With both classifiers SVN_CLASSFIER and RULE_CLASSIFIER, you can use the BASIC_LEXER, CHINESE_LEXER, JAPANESE_LEXER, or KOREAN_MORPH_LEXER lexer. (See "Classifier Types" and "Lexer Types".)

For processing queries, these lexers support the following operators: ABOUT, STEM, AND, NEAR, NOT, OR, and WITHIN.

The thesaural operators (BT*, NT*, PT, RT, SYN, TR, TRSYS, TT, and so on) are supported. However, these operators are expanded using a snapshot of the thesaurus at index time, not when the MATCHES function is entered. This means that if you change your thesaurus after you index, you must re-index your query set.

storage_pref

Specify the storage preference for the index on the queries. Use the storage preference to specify how the index tables are stored. See "Storage Types".

section group

Specify the section group. This parameter does not affect the queries. It applies to sections in the documents to be classified. The following section groups are supported for the CTXRULE index type:

- BASIC SECTION GROUP
- HTML SECTION GROUP
- XML SECTION GROUP



AUTO SECTION GROUP

See "Section Group Types".

CTXRULE does not support special sections. It also does not support NDATA sections.

wordlist_pref

Specifies the wordlist preferences. This is used to enable stemming operations on query terms. See Wordlist Type.

classifier pref

Specifies the classifier preference. See "Classifier Types". You must use the same preference name you specify with CTX CLS.TRAIN.

Example for Creating a CTXRULE Index

See *Oracle Text Application Developer's Guide* for a complete example of using the CTXRULE index type in a document routing application.

Related Topics

CTX_DDL.CREATE_PREFERENCE
CTX_DDL.CREATE_STOPLIST
CTX_DDL.CREATE_SECTION_GROUP
"ALTER INDEX "
"CATSEARCH "

1.6 CREATE SEARCH INDEX

Use the CREATE SEARCH INDEX statement to create a search index for indexing and querying structured, unstructured, or semi-structured data, such as textual, JSON, and XML documents.

Purpose

The SEARCH INDEX is an index type that supports the CONTEXT index functionality along with sharded databases and system-managed partitioning for index storage. Using the CREATE SEARCH INDEX syntax, you can create search indexes on textual, JSON, and XML columns.



Shadow index is not supported for search indexes.

Overview

The CREATE SEARCH INDEX syntax automatically determines the type of search index to create based on the data type of the column, as follows:



Column Data Type	FOR Clause	Syntax Description
Text	FOR TEXT	The CREATE SEARCH INDEX statement on a textual column creates an Oracle Text search index.
		If required, you can explicitly specify the FOR TEXT clause in the CREATE SEARCH INDEX statement to create an Oracle Text search index. If you omit the FOR TEXT clause on a textual column, then the FOR TEXT settings are automatically picked up.
		If a column has the <code>JSON</code> data type, an <code>IS JSON</code> check constraint, or an <code>XMLType</code> data type using TBX, then you can override the settings and create a full-text search index by specifying the <code>FOR TEXT</code> clause.
JSON data type FOR JSON or Column with an IS	FOR JSON	The CREATE SEARCH INDEX statement on a column with the JSON data type or an IS JSON check constraint creates a JSON search index.
JSON check constraint	t	If required, you can explicitly specify the FOR JSON clause in the CREATE SEARCH INDEX statement to create a JSON search index. If you omit the FOR JSON clause and the column has the JSON data type or an IS JSON check constraint, then the FOR JSON settings are automatically picked up.
XMLType data type of TRANSPORTABLE BINARY XML	FOR XML	The CREATE SEARCH INDEX statement on an XMLType column of TRANSPORTABLE BINARY XML (TBX) creates an XML search index. If required, you can explicitly specify the FOR XML clause in the CREATE SEARCH INDEX statement to create an XML search index. If you omit the FOR XML clause on an XMLTYPE column that uses the TBX storage option, then the FOR XML settings are automatically picked up. If you omit the FOR XML clause on an XMLTYPE column but the storage
		option is not TBX, then it creates an Oracle Text index. To create an XML search index, you must ensure that the document is stored as TBX. XML search indexes also support XQuery Full Text search features. You can index XML data that is not stored using the TBX option by creating an XQuery Full Text CONTEXT index. See Oracle XML DB Developer's Guide.

Here is the detailed syntax for each type of search index:

- Syntax for Oracle Text Search Index
- Syntax for JSON Search Index
- Syntax for XML Search Index

Syntax for Oracle Text Search Index

```
CREATE SEARCH INDEX [schema.]index ON [schema.]table(txt_column)
  [ONLINE]
  [FILTER BY filter_column[, filter_column]...]
```



```
[ORDER BY oby_column[desc|asc][, oby_column[desc|asc]]...]
[LOCAL [PARTITION [partition]]
[, PARTITION [partition]])
[PARAMETERS (paramstring)] [PARALLEL n] [UNUSABLE]];
```

ONLINE, FILTER BY, ORDER BY, PARTITION, PARALLEL, and UNUSABLE are described in "Syntax for CONTEXT Index Type".

[schema.]index

Specifies the name of the Oracle Text search index to create.

[schema.]table(index_column)

Specifies the names of table and column to index. index_column is the name of the column on which the index is created.

LOCAL

Creates a local partitioned search index on a partitioned table. The index is partitioned using the partitioning scheme of the base table.

You can partition a table using range, list, hash, interval, range-composite (range, list, and hash), list-composite (range, list, and hash), hash-composite (range, list, and hash), and automatic-list partitioning schemes. You can create a local search index using reference partitioning if the base table of the reference partitioned table is partitioned using any of the supported schemes.



You cannot create a local search index on an interval-composite partitioned table.

Query the views CTX_INDEX_PARTITIONS or CTX_USER_INDEX_PARTITIONS to find out index partition information, such as index partition name and index partition status. The following example shows how to create a text table that is partitioned into three, populate it, and then create a partitioned search index:

```
PROMPT create partitioned table and populate it

CREATE TABLE part_tab (a int, b varchar2(40)) PARTITION BY RANGE(a)
(partition p_tab1 values less than (10),
 partition p_tab2 values less than (20),
 partition p_tab3 values less than (30));

PROMPT create partitioned search index
CREATE SEARCH INDEX part idx ON part tab (b) LOCAL;
```

See Also:

- Creating a Local Partitioned Index
- System Managed Domain Index Supported Schemes in Oracle Database Data Cartridge Developer's Guide



PARAMETERS(paramstring)

Optionally specify indexing parameters in paramstring. You can specify preferences owned by another user using the user.preference notation.

The syntax for paramstring is as follows:

```
paramstring =
'[DATASTORE datastore_pref]
[STORAGE storage_pref]
[MEMORY memsize]
[MAINTENANCE AUTO | MAINTENANCE MANUAL]
[SYNC (MANUAL | EVERY "interval-string" | ON COMMIT)]
[OPTIMIZE (MANUAL | AUTO_DAILY | EVERY "interval-string")]
[STOPLIST stoplist]
[LEXER lexer_pref]
[FILTER filter_pref]
[WORDLIST wordlist_pref]
[SECTION GROUP section group]'
```

Note:

 ${\tt TRANSACTIONAL~and~asynchronous_update~parameters~are~not~supported~for~the~Oracle~Text~search~index~type.}$

DATASTORE datastore pref

Specifies the name of your data store preference. Use the data store preference to specify where your text is stored. See <u>Datastore Types</u>.

The default is direct datastore type.

STORAGE storage pref

Specifies the name of your storage preference for the Oracle Text search index. Use the storage preference to specify how the index tables are stored. See Storage Types.

MEMORY *memsize*

Specifies the amount of run-time memory to use for indexing. The syntax for memsize is:

```
memsize = number[K|M|G]
```

K is for kilobytes, M is for megabytes, and G is for gigabytes.

The value you specify for memsize must be between 1M and the value of MAX_INDEX_MEMORY in the CTX_PARAMETERS view. To specify a memory size larger than the MAX_INDEX_MEMORY, you must reset this parameter with CTX_ADM.SET_PARAMETER to be larger than or equal to memsize.

The default for Oracle Text search index is 500MB.

The memsize parameter specifies the amount of memory Oracle Text uses for indexing before flushing the index to disk. Specifying a large amount memory improves indexing performance because there are fewer I/O operations and improves query performance and maintenance, because there is less fragmentation.

Specifying smaller amounts of memory increases disk I/O and index fragmentation, but might be useful when run-time memory is scarce.

MAINTENANCE AUTO | MAINTENANCE MANUAL

Specifies the maintenance type for synchronization of the Oracle Text search index when there are inserts, updates, or deletes to the base table. The maintenance type specified for an index applies to all index partitions.

You can specify one of the following maintenance types:

Maintenance Type	Description
MAINTENANCE AUTO	This is the default method for synchronizing Oracle Text CONTEXT and search indexes.
	This method sets your index to automatic maintenance, that is, the index is automatically synchronized in the background at optimal intervals.
	You do not need to manually configure a SYNC type or set any synchronization interval. The background mechanism automatically determines the synchronization interval and schedules background SYNC.INDEX operations by tracking the DML queue.
	Note : Shadow indexes do not support automatic maintenance. For a complete list of requirements and restrictions for indexes in an automatic maintenance mode, see <i>Oracle Text Application Developer's Guide</i> .
MAINTENANCE MANUAL	This method sets your index to manual maintenance. This is a non-automatic maintenance (synchronization) mode in which you can specify SYNC types, such as MANUAL, EVERY, or ON COMMIT.

SYNC (MANUAL | EVERY "interval-string" | ON COMMIT)

Specifies the SYNC method for synchronization of the Oracle Text search index when there are inserts, updates, or deletes to the base table.

These SYNC settings are applicable only to the indexes that are set to manual maintenance.



By default, the CONTEXT and search indexes run in an automatic maintenance mode (MAINTENANCE AUTO), which means that your DMLs are automatically synchronized into the index in the background at optimal intervals. Therefore, you do not need to manually configure a SYNC method. However, if required, you can do so if you want to modify the default settings for an index.

You can specify one of the SYNC methods as described in Table 1-10.

Each partition of a locally partitioned index can have its own type of sync (ON COMMIT, EVERY, or MANUAL). The type of sync specified in primary parameter strings applies to all index partitions. Manual sync is the default synchronization method for Oracle Text search indexes. The ON COMMIT sync can be run only serially and must use the same memory size that was specified at index creation.

With automatic (EVERY) synchronization, you can specify memory size and parallel synchronization. You can define repeating schedules in the *interval-string* argument using calendaring syntax values. These values are described in *Oracle Database PL/SQL Packages and Types Reference*.



Syntax:

SYNC [EVERY "interval-string"] MEMORY mem size PARALLEL paradegree

Example:

SYNC [EVERY "freq=secondly;interval=20"] MEMORY 500M PARALLEL 2

The following examples create an Oracle Text search index with automatic (EVERY) synchronization:

Starting every night at 1:00 A.M.:

CREATE SEARCH INDEX nightly_refreshed ON purchase_orders(text_document)
 PARAMETERS('SYNC (EVERY "freq=daily; byhour=1")');

Starting every 5 minutes:

CREATE SEARCH INDEX nightly_refreshed ON purchase_orders(text_document)
PARAMETERS('SYNC (EVERY "freq=minutely; interval=5")');

OPTIMIZE (MANUAL | AUTO_DAILY | EVERY "interval-string)

Specify optimize to enable automatic background index optimization. You can specify any one of the following optimize methods:

OPTIMIZE Type	Description
MANUAL	Provides no automatic optimization. You must manually optimize the index with CTX_DDL.OPTIMIZE_INDEX.
AUTO_DAILY	This is the default value. When you specify OPTIMIZE (AUTO_DAILY) in the CREATE INDEX PARAMETERS string, the continuously running optimize token and optimize full jobs are scheduled. • The optimize token job is scheduled to run every midnight from 12 A.M. to 3 A.M. except on Saturday night, in order to optimize the top 10 most fragmented tokens. Jobs that are not started before 3 A.M. are suspended until 12 A.M. the next day. These suspended jobs are started before the other jobs that are scheduled to run at 12 A.M. the next day. • The optimize full job is scheduled to run weekly from 12 A.M. every Saturday night in order to optimize index tables and clean up \$N. Existing indexes do not have OPTIMIZE (AUTO_DAILY) by default. You need to use ALTER INDEX to enable automatic background index optimization.



OPTIMIZE Type	Description
EVERY "interval- string"	Automatically runs optimize token at a regular interval specified by the value <i>interval-string</i> , which takes the same syntax as the scheduler jobs. Ensure that <i>interval-string</i> is set to a considerable time period so that the previous optimize jobs are complete; otherwise, the optimize job might stop responding. <i>interval-string</i> must be enclosed in double quotes, and any single quote within <i>interval-string</i> must be preceded by the escape character with another single quote.

With $\texttt{AUTO_DAILY} \mid \texttt{EVERY} \mid \texttt{interval-string} \mid \texttt{setting}$, you can specify parallel optimization. That syntax is:

```
... [AUTO DAILY | EVERY "interval-string"] PARALLEL paradegree ...
```

STOPLIST stoplist

Specifies the name of your stoplist. Use stoplist to identify words that are not to be indexed. See CTX_DDL.CREATE_STOPLIST.

The default for Oracle Text search index is CTXSYS.DEFAULT STOPLIST.

LEXER lexer pref

Specifies the name of your lexer or multilexer preference. Use the lexer preference to identify the language of your text and how text is tokenized for indexing. See "Lexer Types". The default is CTXSYS.DEFAULT LEXER.

FILTER filter_pref

Specifies the name of your filter preference. Use the filter preference to specify how to filter formatted documents to plain text or HTML. See "Filter Types".

The default for binary text columns is $\texttt{NULL_FILTER}$. The default for other text columns is AUTO FILTER.

WORDLIST wordlist pref

Specifies the name of your word list preference. Use the word list preference to enable features such as fuzzy, stemming, and prefix indexing for better wild card searching. See "Wordlist Type".

SECTION GROUP section group

Specifies the name of your section group. Use section groups to create sections in structured documents. See "CREATE_SECTION_GROUP" in CTX_DDL Package.

The default value for Oracle Text search index is NULL SECTION GROUP.

Syntax for JSON Search Index

```
CREATE SEARCH INDEX [schema.]index ON [schema.]table(json_column) FOR JSON [LOCAL ([PARTITION [partition]][, PARTITION [partition]])]

PARAMETERS(

[DATAGUIDE ON [CHANGE (ADD_VC | function_name)] | OFF]

[STORAGE storage_pref]

[SEARCH_ON NONE]

[SEARCH_ON (TEXT | TEXT_VALUE[(data_types)] | VALUE[(data_types)] |

TEXT_VALUE_STRING)

[(path_subsetting_clause)]

[MEMORY memsize]

[MAINTENANCE AUTO | MAINTENANCE MANUAL]
```



```
[SYNC (MANUAL | EVERY "interval-string" | ON COMMIT)]
[OPTIMIZE (MANUAL | EVERY "interval-string" | AUTO_DAILY)]
[ASYNCHRONOUS_UPDATE | SYNCHRONOUS_UPDATE]
[POPULATE | NOPOPULATE]
[DATASTORE datastore_pref]
[FILTER filter_pref]
[LEXER lexer_pref]
[WORDLIST wordlist_pref]
)
[PARALLEL N]
[UNUSABLE];
```

If you omit the PARAMETERS clause, then the default values for DATAGUIDE and SEARCH_ON are OFF and TEXT_VALUE respectively. The default synchronization method is MAINTENANCE AUTO. Thus, the index is automatically synchronized in the background, and both text and numeric or date-time ranges are indexed.

Note:

- The SECTION GROUP clause is not required for a JSON search index. You use section groups to define sections in a text column.
- The MULTI_COLUMN_DATASTORE, TRANSACTIONAL, and STOPLIST clauses are not supported for a JSON search index.
- The asynchronous_update, synchronous_update, populate, nopopulate, datastore, filter, lexer, wordlist, parallel, and unusable parameters are described in Syntax for CONTEXT Index Type.

[schema.]index

Specifies the name of the JSON search index to create.

[schema.]table(index column)

Specifies the names of table and column to index. index_column is the name of the column on which the index is created.

The column must have the JSON data type or an IS JSON check constraint.

LOCAL

Creates a local partitioned JSON search index on a partitioned table. The index is partitioned using the partitioning scheme of the base table.

You can partition a table using range, list, hash, interval, range-composite (range, list, and hash), list-composite (range, list, and hash), hash-composite (range, list, and hash), and automatic-list partitioning schemes. You can create a local JSON search index using reference partitioning if the base table of the reference partitioned table is partitioned using any of the supported schemes.

Note:

You cannot create a local JSON search index on an interval-composite partitioned table.

The following example shows how to create a table that is partitioned into three, populate it, and then create a partitioned JSON search index:

```
PROMPT create partitioned table and populate it

CREATE TABLE part_tab (a int, b JSON) PARTITION BY RANGE (a)
(partition p_tabl values less than (10),
partition p_tab2 values less than (20),
partition p_tab3 values less than (30));

PROMPT create partitioned JSON search index
CREATE SEARCH INDEX part_idx ON part_tab (b)
FOR JSON LOCAL;
```

See Also:

- Creating a Local Partitioned Index
- System Managed Domain Index Supported Schemes in Oracle Database Data Cartridge Developer's Guide

DATAGUIDE ON | OFF

Specifies data guide support for a JSON search index. The default behavior is to create a JSON search index without data guide support. If you enable data guide support, then you can also define change-trigger procedures.



You use the DATAGUIDE parameter only for JSON search indexes.

Specify one of the following options:

 ON: Enables data guide support. If you set the value of DATAGUIDE to ON, then you can also define your own PL/SQL procedure or use the predefined change-trigger procedure ADD VC.

ADD VC indicates if virtual columns are created based on the data guide.

function name specifies the function to be executed when the data guide changes.

 OFF: Disables both the data guide support and change-trigger procedures. Provides only general search-index functionality.



You cannot create an index with the SEARCH_ON clause set to NONE when the DATAGUIDE feature is disabled.

See Change Triggers For Data Guide-Enabled Search Index in *Oracle Database JSON Developer's Guide*.

STORAGE storage_pref

Specifies the name of your storage preference for JSON search index. Use the storage preference to specify how index tables are stored. See Storage Types.

SEARCH ON NONE

Does not enable any indexing features, which indicates that the tables used for full-text and range searches are not populated. Only the index data guide is maintained. The index will not be used by any JSON query operators, including JSON_TEXTCONTAINS. For example:

```
CREATE SEARCH INDEX json_idx ON json_tab (jsondoc) FOR JSON PARAMETERS ('SEARCH ON NONE DATAGUIDE ON');
```

SEARCH_ON (TEXT | TEXT_VALUE[(data_types)] | VALUE[(data_types)] | TEXT_VALUE_STRING) [(path_subsetting_clause)]

Specifies the type of data or attributes to be indexed for efficient searching. You can also specify a **path subsetting** clause, as explained in the section that follows.



You can use the SEARCH ON clause only for JSON and XML search indexes.

You can specify one of the following SEARCH ON options:

Option	Description
TEXT	Enables full-text search component, which indicates that only textual data is indexed for full-text search queries. This also includes queries that rely on path information. The index is used for JSON_TEXTCONTAINS predicates and for JSON_VALUE or JSON_EXISTS predicates that manipulate strings when using JSON search index. If your queries involve only full-text search and not stringrange search or numeric search, then you can save some index maintenance time and disk space by specifying this option. Example:
	<pre>CREATE SEARCH INDEX json_idx ON json_tab (jsondoc) FOR JSON PARAMETERS ('SEARCH_ON TEXT');</pre>



Description

VALUE[(data_types)]

Enables range-search component for the specified data types.

This allows the index to be picked up for predicates using relational operators (>, <, ==, >=, <=, !=). A JSON search index that is created with only <code>SEARCH_ON VALUE</code> cannot answer full-text queries by using the <code>JSON_TEXTCONTAINS</code> operator.

Supported data types:

- NUMBER for indexing numeric values.
- TIMESTAMP for indexing date-time values.
- VARCHAR2 for indexing complete string values. The string values are indexed as is without tokenization or other transformations. All the strings that are smaller than or equal to 237 bytes are indexed.

If you do not specify any data type, then the index enables range-search indexing on all supported data types.



The BINARY_DOUBLE data type is allowed only for XML search indexes.

Examples:

VARCHAR2) ');

• This example specifies the default behavior:

```
CREATE SEARCH INDEX json_idx ON json_tab
(jsondoc)
  FOR JSON PARAMETERS ('SEARCH_ON
VALUE');
```

 These examples explicitly specify data types using the VALUE (data_types) syntax:

```
CREATE SEARCH INDEX json_idx ON json_tab
(jsondoc)
  FOR JSON PARAMETERS ('SEARCH_ON
VALUE(TIMESTAMP)');

CREATE SEARCH INDEX json_idx ON json_tab
(jsondoc)
  FOR JSON PARAMETERS ('SEARCH_ON
VALUE(NUMBER,

TIMESTAMP,
```



Description

TEXT_VALUE[(data_ty
pes)]

Enables both the full-text and range-search components for the specified data types.

Supported data types:

- NUMBER for indexing numeric values.
- TIMESTAMP for indexing date-time values.
- VARCHAR2 for indexing complete string values. The string values are indexed as is without tokenization or other transformations. All the strings that are smaller than or equal to 237 bytes are indexed.

If you do not specify any data type, then the index enables full-text search and range-search indexing on ${\tt NUMBER}$ and ${\tt TIMESTAMP}$ data types.

Examples:

• This example specifies the default behavior:

```
CREATE SEARCH INDEX json_idx ON json_tab
(jsondoc)
  FOR JSON PARAMETERS('SEARCH_ON
TEXT_VALUE');
```

 These examples explicitly specify data types using the TEXT VALUE(data types) syntax:

```
CREATE SEARCH INDEX json_idx ON json_tab
(jsondoc)
  FOR JSON PARAMETERS('SEARCH_ON
TEXT_VALUE(NUMBER)');
```

CREATE SEARCH INDEX json_idx ON json_tab
(jsondoc)
 FOR JSON PARAMETERS('SEARCH_ON
TEXT_VALUE(NUMBER,

TIMESTAMP) ');



Option	Description
TEXT_VALUE_STRING	Indicates that text and range-based indexes are created for numeric, date-time, and complete string values. This enables both the full-text and range-search components on the NUMBER, TIMESTAMP, and VARCHAR2 data types. String values are indexed as is without tokenization or other transformations. All the strings that are smaller than or equal to 237 bytes are indexed. Example:
	<pre>CREATE SEARCH INDEX json_idx ON json_tab (jsondoc) FOR JSON PARAMETERS('SEARCH_ON TEXT_VALUE_STRING');</pre>
	Note: For range-search queries, instead of TEXT_VALUE_STRING, Oracle recommends that you use either the VALUE[(data_types)] or TEXT_VALUE[(data_types)] option. Creating an index with TEXT_VALUE(NUMBER, TIMESTAMP, VARCHAR2) is equivalent to TEXT_VALUE_STRING.

path_subsetting_clause

You can use **path subsetting** with <code>SEARCH_ON</code> to identify the fields in a document that you want to include or exclude from indexing. The excluded fields are not indexed, and the JSON search index is not used for them when querying. Filtering out irrelevant paths from documents can reduce the amount of data indexed, thereby minimizing disk space and the index creation or rebuild time.

Syntax for SEARCH ON with path subsetting clause:

```
SEARCH_ON (
  TEXT | TEXT_VALUE [(data_types)] | VALUE [(data_types)] )
  [(INCLUDE | EXCLUDE) '(' paths ')']
```

Note the following:

- You cannot specify both the INCLUDE and EXCLUDE clauses for a single index.
- You can specify a path subsetting clause with SEARCH_ON TEXT, TEXT_VALUE, and VALUE (not with NONE and TEXT VALUE STRING).
- As an alternative to specifying the INCLUDE or EXCLUDE clause, you can use the PATHLIST parameter to specify a list of the paths to be included or excluded. You use PL/SQL subprograms CTX_DDL.CREATE_PATH_LIST and CTX_DDL.ADD_PATH to specify the list of the paths. See CREATE_PATH_LIST.

Specifying any SEARCH_ON clause while there is also a PATHLIST parameter results in an error. Similarly, you cannot specify a PATHLIST parameter for an index that has the Dataguide feature enabled.

Option	Path Subsetting Example
TEXT	 This example creates a JSON search index with path subsetting for full-text and string-equality searches. It indexes only the fields located at paths \$.SpecialInstructions and \$.LineItems.Part.Description of a purchase order document.
	<pre>CREATE SEARCH INDEX json_idx ON json_tab (purchase_order_jsondoc) FOR JSON PARAMETERS ('SEARCH_ON</pre>
	 This example creates a JSON search index with path subsetting for full-text and string-equality searches. It excludes the field located at path \$.User of a purchase order document.
	<pre>CREATE SEARCH INDEX json_idx ON json_tab (purchase_order_jsondoc) FOR JSON PARAMETERS ('SEARCH_ON TEXT EXCLUDE (\$.User');</pre>



Path Subsetting Example

VALUE[(data_types)]

This example creates a JSON search index with path subsetting for full-text and string-equality searches of fields \$.SpecialInstructions and \$.LineItems.Part.Description of a purchase order document. But it also indexes fields \$.PONumber and \$.LineItems.Part.UnitPrice for numeric-value ranges, and fields \$.Reference, \$.User, \$.ShippingInstructions.name, and \$.ShippingInstructions.Address.zinCode

and \$.ShippingInstructions.Address.zipCode
for string-value ranges.

Alternatively, you can create the same index using the PATHLIST parameter, whose value is a named list of the paths to be included, created using PL/SQL subprograms CTX_DDL.create_path_list and CTX_DDL.add_path, as follows:

structions.Address.zipCode)');

```
BEGIN
 CTX DDL.create path list('json pl',
CTX DDL.PATHLIST JSON,
CTX_DDL.PATHLIST INCLUDE);
 CTX DDL.add path('json pl',
            '$.SpecialInstructions');
'TEXT',
 CTX DDL.add path('json pl',
'TEXT',
'$.LineItems.Part.Description');
 CTX DDL.add path('json pl',
'NUMBER',
            '$.PONumber');
 CTX_DDL.add_path('json_pl',
'NUMBER',
'$.LineItems.Part.UnitPrice');
 CTX DDL.add path('json pl',
'VARCHAR2', '$.Reference');
```

Path Subsetting Example

```
CTX_DDL.add_path('json_pl',
'VARCHAR2', '$.User');
  CTX_DDL.add_path('json_pl',
'VARCHAR2',
'$.ShippingInstructions.name');
  CTX_DDL.add_path('json_pl',
'VARCHAR2',
'$.ShippingInstructions.Address.zipCode');
END;
/
CREATE SEARCH INDEX json_idx ON json_tab
(purchase_order_jsondoc)
  FOR JSON PARAMETERS ('PATHLIST
json_pl');
```

 This example creates a JSON search index with path subsetting for numeric-value ranges, where it excludes the field located at path \$.PONumber of a purchase order document.



Path Subsetting Example

TEXT_VALUE[(data_ty •
pes)]

 This example creates a JSON search index with path subsetting for full-text and string-equality searches of fields \$."FieldWithNoQuote" and \$."FieldWith\"Quote\"" of a purchase order document.

Alternatively, you can create the same index using the PATHLIST parameter, whose value is a named list of the paths to be included, created using PL/SQL subprograms CTX_DDL.create_path_list and CTX_DDL.add path, as follows:

```
BEGIN
 CTX DDL.create_path_list('json_pl',
CTX DDL.PATHLIST JSON,
CTX DDL.PATHLIST INCLUDE);
 CTX DDL.add path('json pl',
'TEXT',
            '$.SpecialInstructions');
 CTX DDL.add_path('json pl',
'TEXT',
'$.LineItems.Part.Description');
 CTX DDL.add path('json pl',
'NUMBER',
            '$.PONumber');
 CTX DDL.add path('json pl',
'NUMBER',
'$.LineItems.Part.UnitPrice');
 CTX DDL.add_path('json pl',
'VARCHAR2', '$.Reference');
 CTX DDL.add path('json pl',
'VARCHAR2', '$.User');
 CTX DDL.add path('json pl',
'VARCHAR2',
'$.ShippingInstructions.name');
 CTX DDL.add path('json pl',
'VARCHAR2',
'$.ShippingInstructions.Address.zipCode')
END;
CREATE SEARCH INDEX json idx ON json tab
(purchase order jsondoc)
 FOR JSON PARAMETERS ('PATHLIST
json_pl');
```

Option	Path Subsetting Example
	 This example creates a JSON search index with path subsetting for numeric-value ranges, where it excludes the field located at path \$.PONumber of a purchase order document.
	CREATE SEARCH INDEX json_idx ON json_tab (purchase_order_jsondoc) FOR JSON PARAMETERS ('SEARCH_ON TEXT_VALUE(NUMBER) EXCLUDE (\$.PONumber)');

MEMORY memsize

Specifies the amount of run-time memory to use for indexing. The syntax for memsize is as follows:

```
memsize = number[K|M|G]
```

K is for kilobytes, M is for megabytes, and G is for gigabytes.

The value you specify for memsize must be between 1M and the value of MAX_INDEX_MEMORY in the CTX_PARAMETERS view. To specify a memory size larger than the MAX_INDEX_MEMORY, you must reset this parameter with CTX_ADM.SET_PARAMETER to be larger than or equal to memsize.

The default for JSON search index is the value specified for <code>DEFAULT_INDEX_MEMORY</code> in <code>CTX PARAMETERS</code>.

The memsize parameter specifies the amount of memory Oracle Text uses for indexing before flushing the index to disk. Specifying a large amount memory improves indexing performance because there are fewer I/O operations and improves query performance and maintenance, because there is less fragmentation.

Specifying smaller amounts of memory increases disk I/O and index fragmentation, but might be useful when run-time memory is scarce.

MAINTENANCE AUTO | MAINTENANCE MANUAL

Specifies the maintenance type for synchronization of the JSON search index when there are inserts, updates, or deletes to the base table. The maintenance type specified for an index applies to all index partitions.

You can specify one of the following maintenance types:

Maintenance Type	Description
MAINTENANCE AUTO	This is the default method for synchronizing Oracle Text CONTEXT and search indexes.
	This method sets your index to automatic maintenance, that is, the index is automatically synchronized in the background at optimal intervals.
	You do not need to manually configure a SYNC type or set any synchronization interval. The background mechanism automatically determines the synchronization interval and schedules background SYNC.INDEX operations by tracking the DML queue.
	Note: Shadow indexes do not support automatic maintenance. For a complete list of requirements and restrictions to follow in an automatic maintenance mode, see Oracle Text Application Developer's Guide.
MAINTENANCE MANUAL	This method sets your index to manual maintenance. This is a non-automatic maintenance (synchronization) mode in which you can specify SYNC types, such as MANUAL, EVERY, or ON COMMIT.

SYNC (MANUAL | EVERY "interval-string" | ON COMMIT)

Specifies the SYNC type for synchronization of the JSON search index when there are inserts, updates, or deletes to the base table.

These SYNC settings are applicable only to the indexes that are set to manual maintenance.



By default, the CONTEXT and search indexes run in an automatic maintenance mode (MAINTENANCE AUTO), which means that your DMLs are automatically synchronized into the index in the background at optimal intervals. Therefore, you do not need to manually configure a SYNC method. However, if required, you can do so if you want to modify the default settings for an index.

You can specify one of the SYNC methods as described in Table 1-10.

Each partition of a locally partitioned index can have its own type of sync (ON COMMIT, EVERY, or MANUAL). The type of sync specified in primary parameter strings applies to all index partitions. ON COMMIT sync is the default synchronization method for JSON search indexes. The ON COMMIT sync can be run only serially and must use the same memory size that was specified at index creation.

With automatic (EVERY) synchronization, you can specify memory size and parallel synchronization. You can define repeating schedules in the *interval-string* argument using calendaring syntax values. These values are described in *Oracle Database PL/SQL Packages and Types Reference*.

Syntax:

 ${\tt SYNC \ [EVERY "interval-string"] \ MEMORY \ mem_size \ PARALLEL \ paradegree}$

Example:

SYNC [EVERY "freq=secondly;interval=20"] MEMORY 500M PARALLEL 2

The following examples create a JSON search index with automatic (EVERY) synchronization:

Starting every night at 1:00 A.M.:

CREATE SEARCH INDEX nightly_refreshed ON purchase_orders(json_document)
FOR JSON PARAMETERS('SYNC (EVERY "freq=daily; byhour=1")');

Starting every 5 minutes:

CREATE SEARCH INDEX nightly_refreshed ON purchase_orders(json_document)
FOR JSON PARAMETERS('SYNC (EVERY "freq=minutely; interval=5")');

OPTIMIZE

Specify OPTIMIZE to enable automatic background index optimization. You can specify any of the following OPTIMIZE methods:

OPTIMIZE Type	Description
MANUAL	This is the default value. Provides no automatic optimization. You must manually optimize the index with CTX_DDL.OPTIMIZE_INDEX.
AUTO_DAILY	 When you specify OPTIMIZE (AUTO_DAILY) in the CREATE INDEX PARAMETERS string, the continuously running optimize TOKEN_TYPE and optimize full jobs are scheduled as follows: The optimize TOKEN_TYPE job is scheduled to run every midnight from 12 A.M. to 3 A.M., except on Saturday nights, to optimize SDATA sections in the index. Jobs that are not started before 3 A.M. are suspended until 12 A.M. the next day. These suspended jobs are started before the other jobs that are scheduled to run at 12 A.M. the next day. The optimize full job is scheduled to run weekly from 12 A.M. every Saturday night to optimize index tables and clean up \$N.
EVERY "interval- string"	Automatically runs the optimize TOKEN_TYPE job at a regular interval specified by the value <i>interval-string</i> , which takes the same syntax as scheduler jobs. Ensure that <i>interval-string</i> is set to a considerable time period so that the previous optimize jobs are complete; otherwise, the optimize job might stop responding. <i>interval-string</i> must be enclosed in double quotes, and any single quote within <i>interval-string</i> must be preceded by the escape character with another single quote.



With $\texttt{AUTO_DAILY} \mid \texttt{EVERY} \mid \texttt{interval-string} \mid \texttt{setting}$, you can specify parallel optimization. That syntax is:

```
... [AUTO_DAILY | EVERY "interval-string"] PARALLEL paradegree ...
```

Syntax for XML Search Index

Starting with Oracle Database 23ai, the XML search index provides a simplified syntax for creating XML-enabled indexes. You can create indexes on XML documents that are stored inside an XMLType column or table. This enables you to run textual, path-aware, and range-search queries over XML documents.

```
CREATE SEARCH INDEX [schema.]index ON [schema.]table(xml_column)

FOR XML

[LOCAL]

PARAMETERS(

[SEARCH_ON (TEXT | TEXT_VALUE(data_types) | VALUE(data_types))]

[STORAGE storage_pref]

[PREFIX_NS (prefix_ns_mapping)]

[MEMORY memsize]

[MAINTENANCE AUTO | MAINTENANCE MANUAL]

[SYNC (MANUAL | EVERY "interval-string" | ON COMMIT)]

[OPTIMIZE (MANUAL | EVERY "interval-string" | AUTO_DAILY)]

)

[PARALLEL N]

[UNUSABLE];
```

[schema.]index

Specifies the name of the XML search index to create.

[schema.]table(index_column)

Specifies the names of table and column to index. index_column is the name of the column on which the index is created.

You can create the index only on an XMLType column that stores documents using the TRANSPORTABLE BINARY XML (TBX) storage option.

LOCAL

Creates a local partitioned XML search index on a partitioned table. The index is partitioned using the partitioning scheme of the base table.

You can partition a table using range, list, hash, interval, range-composite (range, list, and hash), list-composite (range, list, and hash), hash-composite (range, list, and hash), and automatic-list partitioning schemes. You can create a local XML search index using reference partitioning if the base table of the reference partitioned table is partitioned using any of the supported schemes.



You cannot create a local XML search index on an interval-composite partitioned table

The following example shows how to create a table that is partitioned into three, populate it, and then create a partitioned XML search index:

```
PROMPT create partitioned table and populate it

CREATE TABLE part_tab (a int, b SYS.XMLType) XMLTYPE b STORE AS

TRANSPORTABLE BINARY XML PARTITION BY RANGE (a)

(partition p_tab1 values less than (10),

partition p_tab2 values less than (20),

partition p_tab3 values less than (30));

PROMPT create partitioned XML search index

CREATE SEARCH INDEX part_idx ON part_tab (b)

FOR XML PARAMETERS ('SEARCH ON TEXT') LOCAL;
```

STORAGE storage pref

Specifies the name of your storage preference for XML search index. Use the storage preference to specify how index tables are stored. See "Storage Types". If you do not specify a storage preference, then the default storage preference (CTXSYS.XQFT MEDIUM) is used.

SEARCH_ON (TEXT | TEXT_VALUE(*data_types***) | VALUE(***data_types***))**Specifies the type of data or attributes to be indexed for efficient searching.



You can use the SEARCH ON clause only for JSON and XML search indexes.

You can specify one of the following SEARCH ON options:

Option	Description
TEXT	Enables full-text search component, which indicates that only textual data is indexed for full-text search queries. This also includes queries that rely on path information. The index is used for XMLEXISTS predicates that references the XQuery Full Text operators and clauses. If your queries involve only full-text search and not stringrange search or numeric search, then you can save some index maintenance time and disk space by specifying this option. For example:
	<pre>CREATE SEARCH INDEX ex_xml_idx ON ex_tab (xmldoc) FOR XML PARAMETERS ('SEARCH_ON TEXT');</pre>

Description

VALUE(data types)

Enables range-search component for the specified data types.

This allows the index to be picked up for predicates using relational operators (>, <, ==, >=, <=, !=). An XML search index that only has the <code>SEARCH_ON VALUE</code> component enabled cannot answer full-text queries, if XQuery Full Text operators are present in an <code>XMLEXISTS</code> predicate.

You must specify one or more data types:

- BINARY_DOUBLE and NUMBER for indexing numeric values.
- TIMESTAMP for indexing date-time values.
- VARCHAR2 for indexing complete string values. The string values are indexed as is without tokenization or other transformations. All the strings that are smaller than or equal to 237 bytes are indexed.

For example:

VARCHAR2) ');

```
CREATE SEARCH INDEX ex_xml_idx ON ex_tab
(xmldoc)
   FOR XML PARAMETERS ('SEARCH_ON
VALUE(NUMBER)');

CREATE SEARCH INDEX ex_xml_idx ON ex_tab
(xmldoc)
   FOR XML PARAMETERS ('SEARCH_ON
VALUE(BINARY_DOUBLE,

NUMBER,
```



Option	Description
TEXT_VALUE(data_typ es)	Enables both the full-text and range-search components for the specified data types. For range-search queries, you must specify one or more data types, such as NUMBER (for indexing numeric values) and TIMESTAMP (for indexing date-time values). For example:
	<pre>CREATE SEARCH INDEX ex_xml_idx ON ex_tab (xmldoc) FOR XML PARAMETERS('SEARCH_ON TEXT_VALUE(TIMESTAMP)');</pre>
	CREATE SEARCH INDEX ex_xml_idx ON ex_tab (xmldoc) FOR XML PARAMETERS('SEARCH_ON TEXT_VALUE(NUMBER,
	TIMESTAMP)');

Note:

You cannot use <code>SEARCH_ON NONE</code> and <code>SEARCH_ON TEXT_VALUE_STRING</code> for an XML search index.

You must explicitly specify a data type with the ${\tt TEXT_VALUE}$ and ${\tt VALUE}$ options for an XML search index, otherwise the statement will result in an error.

PREFIX_NS (prefix_ns_mapping)

Specifies prefix-namespace mapping for an XML search index.

An XMLExists query can include XML namespace declarations. While creating the search index, you can separately store qualified names belonging to different XML namespaces. A prefix-namespace mapping uses this syntax:

```
xmlns:local name="URI string"
```

xmlns is the default XML namespace declaration attribute. The URI_string value is not mandatory. You can provide an empty string enclosed in double quotation marks. You can also specify a qualified-name with the xmlns prefix. If you do not specify a prefix-namespace mapping, then xmlns is used.

For example:

```
PREFIX_NS (xmlns="example.com" xmlns:pfx="www.example1.com"
xmlns:pfx2="example2.com"));
```



You use the PREFIX NS clause only for XML search indexes.

You cannot use ALTER INDEX to modify a prefix-namespace mapping specification.

MEMORY memsize

Specifies the amount of run-time memory to use for indexing. The syntax for memsize is as follows:

memsize = number[K|M|G]

K is for kilobytes, M is for megabytes, and G is for gigabytes.

The value you specify for memsize must be between 1M and the value of MAX_INDEX_MEMORY in the CTX_PARAMETERS view. To specify a memory size larger than the MAX_INDEX_MEMORY, you must reset this parameter with CTX_ADM.SET_PARAMETER to be larger than or equal to memsize.

The memsize parameter specifies the amount of memory Oracle Text uses for indexing before flushing the index to disk. Specifying a large amount memory improves indexing performance because there are fewer I/O operations and improves query performance and maintenance, because there is less fragmentation.

Specifying smaller amounts of memory increases disk I/O and index fragmentation, but might be useful when run-time memory is scarce.

MAINTENANCE AUTO | MAINTENANCE MANUAL

Specifies the maintenance type for synchronization of the XML search index when there are inserts, updates, or deletes to the base table. The maintenance type specified for an index applies to all index partitions.

You can specify one of the following maintenance types:

Maintenance Type	Description
MAINTENANCE AUTO	This is the default method for synchronizing Oracle Text CONTEXT and search indexes.
	This method sets your index to automatic maintenance, that is, the index is automatically synchronized in the background at optimal intervals.
	You do not need to manually configure a SYNC type or set any synchronization interval. The background mechanism automatically determines the synchronization interval and schedules background SYNC.INDEX operations by tracking the DML queue. Note: Shadow indexes do not support automatic
	maintenance. For a complete list of requirements and restrictions to follow in an automatic maintenance mode, see <i>Oracle Text Application Developer's Guide</i> .
MAINTENANCE MANUAL	This method sets your index to manual maintenance. This is a non-automatic maintenance (synchronization) mode in which you can specify SYNC types, such as MANUAL, EVERY, or ON COMMIT.

SYNC (MANUAL | EVERY "interval-string" | ON COMMIT)

Specifies the SYNC method for synchronization of the XML search index when there are inserts, updates, or deletes to the base table.

These SYNC settings are applicable only to the indexes that are set to manual maintenance.



By default, the CONTEXT and search indexes run in an automatic maintenance mode (MAINTENANCE AUTO), which means that your DMLs are automatically synchronized into the index in the background at optimal intervals. Therefore, you do not need to manually configure a SYNC method. However, if required, you can do so if you want to modify the default settings for an index.

You can specify one of the SYNC methods as described in Table 1-10.

Each partition of a locally partitioned index can have its own type of sync (ON COMMIT, EVERY, or MANUAL). The type of sync specified in primary parameter strings applies to all index partitions. ON COMMIT is the default synchronization method for XML search indexes. The ON COMMIT sync can be run only serially and must use the same memory size that was specified at index creation.

With automatic (EVERY) synchronization, you can specify memory size and parallel synchronization. You can define repeating schedules in the *interval-string* argument using calendaring syntax values. These values are described in *Oracle Database PL/SQL Packages* and Types Reference.

Syntax:

SYNC [EVERY "interval-string"] MEMORY mem size PARALLEL paradegree

Example:

SYNC [EVERY "freq=secondly;interval=20"] MEMORY 500M PARALLEL 2

The following examples create an XML search index with automatic (EVERY) synchronization:

Starting every night at 1:00 A.M.:

```
CREATE SEARCH INDEX nightly_refreshed ON purchase_orders(xml_document)
FOR XML PARAMETERS('SYNC (EVERY "freq=daily; byhour=1")');
```

Starting every 5 minutes:

```
CREATE SEARCH INDEX nightly_refreshed ON purchase_orders(xml_document)
FOR XML PARAMETERS('SYNC (EVERY "freq=minutely; interval=5")');
```

OPTIMIZE

Specify OPTIMIZE to enable automatic background index optimization. You can specify one of the following OPTIMIZE methods:

OPTIMIZE Type	Description
MANUAL	This is the default value.
	Provides no automatic optimization. You must manually
	optimize the index with CTX_DDL.OPTIMIZE_INDEX.



OPTIMIZE Type	Description
AUTO_DAILY	When you specify OPTIMIZE (AUTO_DAILY) in the CREATE INDEX PARAMETERS string, the continuously running optimize TOKEN_TYPE and optimize full jobs are scheduled as follows: • The optimize TOKEN_TYPE job is scheduled to run every midnight from 12 A.M. to 3 A.M., except on Saturday nights, to optimize SDATA sections in the index. Jobs that are not started before 3 A.M. are suspended until 12 A.M. the next day. These suspended jobs are started before the other jobs that are scheduled to run at 12 A.M. the next day. • The optimize full job is scheduled to run weekly from 12 A.M. every Saturday night to optimize index tables and clean up \$N.
EVERY "interval- string"	Automatically runs the optimize <code>TOKEN_TYPE</code> job at a regular interval specified by the value <code>interval-string</code> , which takes the same syntax as scheduler jobs. Ensure that <code>interval-string</code> is set to a considerable time period so that the previous optimize jobs are complete; otherwise, the optimize job might stop responding. <code>interval-string</code> must be enclosed in double quotes, and any single quote within <code>interval-string</code> must be preceded by the escape character with another single quote.

With $\texttt{AUTO_DAILY} \mid \texttt{EVERY} \text{ "} interval-string"$ setting, you can specify parallel optimization. That syntax is:

```
... [AUTO_DAILY | EVERY "interval-string"] PARALLEL paradegree ...
```

Related Topics

- Oracle Database Administrator's Guide
- Oracle Database JSON Developer's Guide

1.7 CREATE HYBRID VECTOR INDEX

Use the CREATE HYBRID VECTOR INDEX SQL statement to create a hybrid vector index, which allows you to index and query documents using a combination of full-text search and vector similarity search.

Purpose

To create a class of specialized Domain Index called a hybrid vector index.

A **hybrid vector index** is an Oracle Text SEARCH INDEX type that combines the existing Oracle Text indexing data structures and vector indexing data structures into one unified structure. It is a single domain index that stores both text fields and vector fields for a document. Both text search and similarity search are performed on tokenized terms and vectors respectively. The two search results are combined and scored to return a unified result set.

The purpose of a hybrid vector index is to enhance search relevance of an Oracle Text index by allowing users to search by both *vectors* and *keywords*. By integrating traditional keyword-

based text search with vector-based similarity search, you can improve the overall search experience and provide users with more accurate information.

Usage Notes

To create a hybrid vector index, you can provide minimal information such as:

- table or column on which you want to create the index
- in-database ONNX embedding model for generating embeddings

For cases where multiple columns or tables need to be indexed together, you can specify the MULTI COLUMN DATASTORE or USER DATASTORE preference.

All other indexing parameters are predefined to facilitate the indexing of documents without requiring you to be an expert in any text processing, chunking, or embedding strategies. If required, you can modify the predefined parameters using:

- Vector search preferences for the vector index part of the index
- Text search preferences for the text index part of the index
- Index maintenance preferences for DML operations on the combined index

For detailed information on the creation process of a hybrid vector index or in general about what hybrid vector indexes are, see Understand Hybrid Vector Indexes.



There are some key points to note when creating and using hybrid vector indexes. See Guidelines and Restrictions for Hybrid Vector Indexes.

Syntax

```
CREATE HYBRID VECTOR INDEX [schema.]index_name ON
[schema.]table_name(column_name)

PARAMETERS ('paramstring')

[FILTER BY filter_column[, filter_column]...]

[ORDER BY oby_column[desc|asc][, oby_column[desc|asc]]...]

[PARALLEL n];
```

Here is an example DDL specified with only the minimum required parameters.

```
CREATE HYBRID VECTOR INDEX my_hybrid_idx on
  doc_table(text_column)
  PARAMETERS('MODEL my_embed_model');
```

More comprehensive examples are given at the end of this section.

Let us explore all the required and optional indexing parameters:

[schema.]index name

Specify the name of the hybrid vector index to create.



[schema.]table_name(column_name)

Specify the name of the table and column on which you want to create the hybrid vector index. You can create a hybrid vector index on one or more text columns with VARCHAR2, CLOB, and BLOB data types.



You cannot create hybrid vector indexes on a text column that uses the ${\tt IS}$ ${\tt JSON}$ check constraint.

Because the system can index most document formats, including HTML, PDF, Microsoft Word, and plain text, you can load a supported type into the text column. For a complete list, see Supported Document Formats.

For cases where multiple columns or tables need to be indexed together, specify a datastore preference (described later in Text search preferences).

PARAMETERS (paramstring)

Specify preferences in paramstring:

Vector Search Preferences:

Configures the "vector index" part of a hybrid vector index, pertaining to processing input for vector search.

Note:

You can either pass a minimal set of parameters (the required MODEL and the optional VECTOR_IDXTYPE parameters) directly in the PARAMETERS clause or use a vectorizer preference to specify a complete set of vector search parameters. You cannot use both (directly set parameters along with vectorizer) in the PARAMETERS clause.

With MODEL and VECTOR IDXTYPE directly specified:

```
CREATE HYBRID VECTOR INDEX [schema.]index_name ON
[schema.]table_name(column_name)

PARAMETERS ('MODEL <model_name>
[VECTOR_IDXTYPE <vector_index_type>]')

[FILTER BY filter_column[, filter_column]...]

[ORDER BY oby_column[desc|asc][, oby_column[desc|asc]]...]

[PARALLEL n];
```

Here, MODEL specifies the vector embedding model that you import into the database for generating vector embeddings on your input data.



Currently, only ONNX in-database embedding models are supported.

VECTOR_IDXTYPE specifies the type of vector index to create, such as IVF (default) for the Inverted File Flat (IVF) vector index and HNSW for the Hierarchical Navigable Small World (HNSW) vector index. If you omit this parameter, then the IVF vector index is created by default.

Creating a LOCAL index on an Hybrid Vector Index is supported when the underlying index type is IVF. An example is shown below:



Caution:

Creating a LOCAL index on Hybrid Vector Index when the underlying index_type is HNSW, would throw an error before starting any document processing (early failure).

With the vectorizer preference:

A vectorizer preference is a JSON object that collectively holds all indexing parameters related to chunking (UTL_TO_CHUNKS or VECTOR_CHUNKS), embedding (UTL_TO_EMBEDDING, UTL_TO_EMBEDDINGS, or VECTOR_EMBEDDING), and vector index (distance, accuracy, or vector_idxtype).

You use the <code>DBMS_VECTOR_CHAIN.CREATE_PREFERENCE</code> PL/SQL function to create a vectorizer preference. To create a vectorizer preference, see <code>DBMS_VECTOR_CHAIN.CREATE_PREFERENCE</code>.

After creating a vectorizer preference, you can use the VECTORIZER parameter to pass the preference name here. For example:



```
}'
));
end;

CREATE HYBRID VECTOR INDEX my_hybrid_idx on
doc_table(text_column)
   parameters('VECTORIZER my vectorizer pref');
```

Text Search Preferences:

Configures the "Oracle Text index" part of a hybrid vector index, pertaining to processing input for keyword search.

These parameters define the text processing and tokenization stages of a hybrid vector indexing pipeline. All these are the same set of parameters that you provide when working with Oracle Text indexes alone.

```
[DATASTORE datastore_pref]
[STORAGE storage_pref]
[MEMORY memsize]
[STOPLIST stoplist]
[LEXER lexer_pref]
[FILTER filter_pref]
[WORDLIST wordlist_pref]
[SECTION GROUP section group]
```

DATASTORE datastore_pref

Specify the name of your datastore preference. Use the datastore preference to specify the local or remote location where your source files are stored.

If you want to index multiple columns or tables together, see MULTI_COLUMN_DATASTORE and USER_DATASTORE.

For a complete list of all datastore preferences, see Datastore Types.

Default: DIRECT DATASTORE

STORAGE storage_pref

Specify the name of your storage preference for an Oracle Text search index. Use the storage preference to specify how the index tables are stored. See Storage Types.

MEMORY memsize

Specify the amount of run-time memory to use for indexing.

```
memsize = number[K|M|G]
```

K is for kilobytes, M is for megabytes, and G is for gigabytes.

The value you specify for memsize must be between 1M and the value of MAX_INDEX_MEMORY in the CTX_PARAMETERS view. To specify a memory size larger than the MAX_INDEX_MEMORY, you must reset this parameter with CTX_ADM.SET_PARAMETER to be larger than or equal to memsize. See CTX_ADM.SET_PARAMETER.

The default for Oracle Text search index is 500 MB.



The memsize parameter specifies the amount of memory Oracle Text uses for indexing before flushing the index to disk. Specifying a large amount memory improves indexing performance because there are fewer I/O operations and improves query performance and maintenance, because there is less fragmentation.

Specifying smaller amounts of memory increases disk I/O and index fragmentation, but might be useful when run-time memory is scarce.

STOPLIST stoplist

Specify the name of your stoplist. Use stoplist to identify words that are not to be indexed. See CTX_DDL.CREATE_STOPLIST.

Default: CTXSYS.DEFAULT STOPLIST

LEXER lexer pref

Specify the name of your lexer or multilexer preference. Use the lexer preference to identify the language of your text and how text is tokenized for indexing. See Lexer Types.

Default: CTXSYS.DEFAULT LEXER

FILTER filter_pref

Specify the name of your filter preference. Use the filter preference to specify how to filter formatted documents to plain text. See Filter Types.

The default for binary text columns is <code>NULL_FILTER</code>. The default for other text columns is <code>AUTO FILTER</code>.

WORDLIST wordlist pref

Specify the name of your wordlist preference. Use the wordlist preference to enable features such as fuzzy, stemming, and prefix indexing for better wildcard searching. See Wordlist Type.

SECTION GROUP section_group

Specify the name of your section group. Use section groups to create sections in structured documents. See CTX_DDL.CREATE_SECTION_GROUP.

Default: NULL SECTION GROUP

Index Maintenance Preferences:

Configures the DML operations on the entire hybrid vector index, that is, how to synchronize and optimize the index.

Because a hybrid vector index is basically an Oracle Text search index type, so all maintenance-specific capabilities of an Oracle Text index are applicable.

```
[MAINTENANCE AUTO | MAINTENANCE MANUAL]
[SYNC (MANUAL | EVERY "interval-string" | ON COMMIT)]
[OPTIMIZE (MANUAL | AUTO_DAILY | EVERY "interval-string")]
```

MAINTENANCE AUTO | MAINTENANCE MANUAL

Specify the maintenance type for synchronization of a hybrid vector index when there are inserts, updates, or deletes to the base table. The maintenance type specified for an index applies to all index partitions.



You can specify one of the following maintenance types:

Maintenance Type	Description
MAINTENANCE AUTO	This method sets your index to automatic maintenance, that is, the index is automatically synchronized in the background at optimal intervals. You do not need to manually configure a SYNC type or set any synchronization interval. The background mechanism automatically determines the synchronization interval and schedules background SYNC.INDEX operations by tracking the DML queue.
MAINTENANCE MANUAL	This method sets your index to manual maintenance. This is a non-automatic maintenance (synchronization) mode in which you can specify SYNC types, such as MANUAL, EVERY, or ON COMMIT.

SYNC (MANUAL | EVERY "interval-string" | ON COMMIT)

Specify the $\tt SYNC$ type for synchronization of a hybrid vector index when there are inserts, updates, or deletes to the base table. These $\tt SYNC$ settings are applicable only to the indexes that are set to manual maintenance.



By default, a hybrid vector index runs in an automatic maintenance mode (MAINTENANCE AUTO), which means that your DMLs are automatically synchronized into the index in the background at optimal intervals. Therefore, you do not need to manually configure a SYNC type for maintaining a hybrid vector index. However, if required, you can do so if you want to modify the default settings for an index.

You can specify one of the SYNC methods:

SYNC Type	Description
MANUAL	With this method, automatic synchronization is not provided. You must manually synchronize the index using CTX_DDL.SYNC_INDEX.
EVERY interval-string	Automatically synchronize the index at a regular interval specified by the value of <i>interval-string</i> , which takes the same syntax as that for scheduler jobs. Automatic synchronization using EVERY requires that the index creator have CREATE JOB privileges. Ensure that <i>interval-string</i> is set to a considerable time period so that any previous synchronization jobs will have completed. Otherwise, the synchronization job may stop responding. The <i>interval-string</i> argument must be enclosed in double quotation marks (" ").



SYNC Type	Description
ON COMMIT	Synchronize the index immediately after a commit. The commit does not return until the sync is complete. The operation uses the memory specified with the memory parameter. This sync type works best when the STAGE_ITAB index option is enabled, otherwise it causes significant fragmentation of the main index, requiring frequent OPTIMIZE calls.

With automatic (EVERY) synchronization, you can specify memory size and parallel synchronization. You can define repeating schedules in the *interval-string* argument using calendaring syntax values. These values are described in *Oracle Database PL/SQL Packages and Types Reference*.

Syntax:

SYNC [EVERY "interval-string"] MEMORY mem size PARALLEL paradegree

For example, to sync the index at an interval of 20 seconds:

SYNC [EVERY "freq=secondly;interval=20"] MEMORY 500M PARALLEL 2

OPTIMIZE (MANUAL | AUTO_DAILY | EVERY "interval-string)

Specify OPTIMIZE to enable automatic background index optimization of a hybrid vector index. You can specify any one of the following OPTIMIZE methods:

OPTIMIZE Type	Description
MANUAL	Provides no automatic optimization. You must manually optimize the index with CTX_DDL.OPTIMIZE_INDEX.
AUTO_DAILY	This is the default setting. With OPTIMIZE (AUTO_DAILY), the optimize FULL job is scheduled to run midnight from 12 A.M. local time everyday.
EVERY "interval- string"	Automatically runs optimize token at a regular interval specified by the value <i>interval-string</i> , which takes the same syntax as the scheduler jobs. Ensure that <i>interval-string</i> is set to a considerable time period so that the previous optimize jobs are complete; otherwise, the optimize job might stop responding. <i>interval-string</i> must be enclosed in double quotes, and any single quote within <i>interval-string</i> must be preceded by the escape character with another single quote.

With ${\tt AUTO_DAILY} \mid {\tt EVERY}$ "interval-string" setting, you can specify parallel optimization.

Syntax:

```
OPTIMIZE [AUTO DAILY | EVERY "interval-string"] PARALLEL paradegree ...
```

For example, to optimize the index at an interval of 20 minutes:

```
OPTIMIZE [EVERY "freq=minutely;interval=20"] PARALLEL 2
```

FILTER BY filter column

Specify the structured indexed column on which a range or equality predicate in the WHERE clause of a mixed query will operate. You can specify one or more structured columns for filter_column, on which the relational predicates are expected to be specified along with the CONTAINS() predicate in a guery.

You can use these relational operators:

```
<, <=, =, >=, >, between, and LIKE (for VARCHAR2)
```

 These columns can only be of CHAR, NUMBER, DATE, VARCHAR2, or RAW type. Additionally, CHAR, VARCHAR2 and VARCHAR2 types are supported only if the maximum length is specified and does not exceed 249 bytes.

If the maximum length of a CHAR or VARCHAR2 column is specified in characters, for example, VARCHAR2 (50 CHAR), then it cannot exceed FLOOR (249/max_char_width), where max char width is the maximum width of any character in the database character set.

For example, the maximum specified column length cannot exceed 62 characters, if the database character set is AL32UTF8. The ADT attributes of supported types (CHAR, NUMBER, DATE, VARCHAR2, or RAW) are also allowed.

An error is raised for all other data types. Expressions, for example, func (cola), and virtual columns are not allowed.

- txt column is allowed in the FILTER BY column list.
- DML operations on FILTER BY columns are always transactional.

ORDER BY oby column[desclasc]

Specify one or more structured indexed columns by which you want to sort query results. You can specify a list of structured *oby_columns*. These columns can only be of CHAR, NUMBER, DATE, VARCHAR2, or RAW type. VARCHAR2 and RAW columns longer than 249 bytes are truncated to the first 249 bytes. Expressions, for example func(cola), and virtual columns are not allowed. The order of the specified columns matters. The ORDER BY clause in a query can contain:

- The entire ordered ORDER BY columns
- Only the prefix of the ordered ORDER BY columns
- The score followed by the prefix of the ordered ORDER BY columns

DESC sorts the results in a descending order (from highest to lowest), while ASC (default) sorts the results in an ascending order (from lowest to highest).

[PARALLEL n]

Parallel indexing can improve index performance when you have multiple CPUs. To create an index in parallel, use the PARALLEL clause with a parallel degree.

Optionally specifies the parallel degree for parallel indexing. The actual degree of parallelism might be smaller depending on your resources. You can use this parameter on nonpartitioned



tables. However, creating a nonpartitioned index in parallel does not turn on parallel query processing. Parallel indexing is supported for creating a local partitioned index. The indexing memory size specified in the parameter clause applies to each parallel worker. For example, if indexing memory size is specified in the parameter clause as 500M and parallel degree is specified as 2, then you must ensure that there is at least 1GB of memory available for indexing.

Examples

With vector search preferences directly specified:

In this example, only the required parameter model is specified in the PARAMETERS clause:

```
CREATE HYBRID VECTOR INDEX my_hybrid_idx on
  doc_table(text_column)
  parameters('MODEL my_doc_model');
```

In this example, both the parameters model and vector idxtype are specified:

With vector search preferences specified using VECTORIZER:

In this example, the vectorizer parameter is used in the PARAMETERS clause to specify the my vectorizer spec preference:

With text search and vector search preferences directly specified:

In this example, only the required Vector Search parameter MODEL is specified in the PARAMETERS clause. Text Search parameters are also specified:

```
STOPLIST my_stoplist

LEXER my_lexer')

ORDER BY docid asc;
```

 With text search and index maintenance preferences directly specified and vector search preferences specified using VECTORIZER:

In this example, the VECTORIZER parameter is used to specify the my_vectorizer_spec preference that holds vector search parameters. All the Text Search and Index Maintenance preferences are directly specified.

```
begin
  DBMS VECTOR CHAIN.CREATE PREFERENCE (
    'my vectorizer spec',
     dbms vector chain.vectorizer,
    json('{
             "vector idxtype" : "hnsw",
            "model" : "my_doc_model",
"by" : "words",
            "by"
"max"
            "max" : 100,
"overlap" : 10,
"split" : "recursively"
        ));
end;
CREATE HYBRID VECTOR INDEX my hybrid idx on
  doc table(text column)
 parameters('VECTORIZER my_vectorizer_spec
              DATASTORE my datastore
              STORAGE my storage
              MEMORY 128M
              MAINTENANCE AUTO
              OPTIMIZE AUTO DAILY
              STOPLIST my stoplist
              LEXER my lexer
              FILTER my filter
              WORDLIST my wordlist
              SECTION GROUP my section group')
  FILTER BY category, author
  ORDER BY score(10), score(20) desc
  PARALLEL 3;
```

Related Topics

- Perform Hybrid Search
- Query Hybrid Vector Indexes End-to-End Example

1.8 DROP INDEX



This section describes the DROP INDEX statement as it pertains to dropping a Text domain index.

For a complete description of the DROP INDEX statement, see *Oracle Database SQL Language Reference*.

Purpose

Use DROP INDEX to drop a specified Text index.

Syntax

DROP INDEX [schema.] index [force];

[force]

Optionally forces the index to be dropped. Use the force option when Oracle Text cannot determine the state of the index, such as when an indexing operation fails.

Oracle recommends against using this option by default. Use it only when a regular call to DROP INDEX fails.

Example

The following example drops an index named <code>doc_index</code> in the current user's database schema:

DROP INDEX doc_index;

Related Topics

"ALTER INDEX "

"CREATE INDEX"

1.9 MATCHES

Use the MATCHES operator to find all rows in a query table that match a given document. The document must be a plain text, HTML, or XML document.

The MATCHES operator also supports database links. You can identify a remote table or materialized view by appending <code>@dblink</code> to the end of its name. The <code>dblink</code> must be a complete or partial name for a database link to the database containing the remote table or materialized view. (Querying of remote views is not supported.)

This operator requires a CTXRULE index on your set of queries.

When the SVM_CLASSIFIER classifier type is used, MATCHES returns a score in the range 0 to 100; a higher number indicates a greater confidence in the match. Use the label parameter and MATCH_SCORE to obtain this number. Then use the matching score to apply a category-specific threshold to a particular category.



If the SVM_CLASSIFIER type is not used, then this operator returns either 100 (the document matches the criteria) or 0 (the document does not match).

Limitation

If the optimizer chooses to use the functional query invocation with a MATCHES query, your query will fail.

Syntax

```
MATCHES(
[schema.]column,
document VARCHAR2 or CLOB
[,label INTEGER])
RETURN NUMBER;
```

column

Specifies the column containing the indexed query set.

document

Specifies the document to be classified. The document can be plain text, HTML, or XML. Binary formats are not supported.

label

Optionally specifies the label that identifies the score generated by the MATCHES operator. Use this label with MATCH_SCORE.

Matches Example

The following example creates a table querytable, and populates it with classification names and associated rules. It then creates a CTXRULE index.

The example enters the MATCHES query with a document string to be classified. The SELECT statement returns all rows (queries) that are satisfied by the document:

Related Topics

```
"MATCH_SCORE"

"Syntax for CTXRULE Index Type"

CTX_CLS.TRAIN
```



Oracle Text Application Developer's Guide contains extended examples of simple and supervised classification, which make use of the MATCHES operator.

1.10 MATCH_SCORE

Use the MATCH SCORE operator in a statement to return scores produced by a MATCHES query.

The MATCH_SCORE operator also supports database links. You can identify a remote table or materialized view by appending <code>@dblink</code> to the end of its name. The <code>dblink</code> must be a complete or partial name for a database link to the database containing the remote table or materialized view. (Querying of remote views is not supported.)

When the SVM_CLASSIFIER classifier type is used, this operator returns a score in the range 0 to 100. Use the matching score to apply a category-specific threshold to a particular category.

If the SVM_CLASSIFIER classifier is not used, then this operator returns either 100 (the document matches the criteria) or 0 (the document does not match).

Syntax

MATCH SCORE (label NUMBER)

label

Specifies a number to identify the score produced by the query. Use this number to identify the MATCHES clause that returns this score.

Example

To get the matching score, use:

```
select cat_id, match_score(1) from training_result where matches(profile,
text,1)>0;
```

Related Topics

"MATCHES"

1.11 SCORE

Use the SCORE operator in SELECT statements to return the score values produced by CONTAINS and JSON TEXTCONTAINS queries.

The SCORE operator can be used in a SELECT, ORDER BY, or GROUP BY clause.

The SCORE operator also supports database links. You can identify a remote table or materialized view by appending <code>@dblink</code> to the end of its name. The <code>dblink</code> must be a complete or partial name for a database link to the database containing the remote table or materialized view. (Querying of remote views is not supported.)

Syntax

SCORE (label NUMBER)

Here, label specifies a number to identify the score produced by the query. Use this number to identify the CONTAINS clause that returns this score.



Notes

For nested queries, you must specify an alias to avoid errors. For example, here an alias "s" is used in the inner SELECT query to identify the outer SELECT query:

Examples

With a single CONTAINS clause:

When the SCORE operator is called (for example, in a SELECT clause), the CONTAINS clause must reference the score label value as in the following example:

```
SELECT SCORE(1), title from newsindex
    WHERE CONTAINS(text, 'oracle', 1) > 0
    ORDER BY SCORE(1) DESC;
```

With multiple CONTAINS clauses:

Assume that a news database stores and indexes the title and body of news articles separately. The following query returns all the documents that include the words *Oracle* in their title and *java* in their body. The articles are sorted by the scores for the first CONTAINS (*Oracle*) and then by the scores for the second CONTAINS (*java*).

```
SELECT title, body, SCORE(10), SCORE(20)
    FROM news
    WHERE CONTAINS (news.title, 'Oracle', 10) > 0 OR
    CONTAINS (news.body, 'java', 20) > 0
    ORDER BY SCORE(10), SCORE(20);
```

With a single JSON_TEXTCONTAINS clause:

This query selects the PO numbers of purchase orders whose descriptions contain the text run. It orders the results by relevance using an optional scoring-label argument. The query returns also the relevance score for each purchase order.

The scoring label passed to <code>json_textcontains</code> must be the same as the label used with <code>score</code>. In this case the label is 1.

Results (some elided):

The first 17 purchase orders listed have score 18; the remaining 85 purchase orders have score 9. The former group match pattern run better than the latter (they match it twice per purchase order instead of once):

```
PONUMBER SCORE (1)
```



```
1 18

9958 18

...

1388 18

36 9

22 9

...

8637 9

102 rows selected.
```

Related Topics

CONTAINS

Use the ${\tt CONTAINS}$ operator in the where clause of a <code>SELECT</code> statement to specify the query expression for a Text query.

- JSON_TEXTCONTAINS
- The Oracle Text Scoring Algorithm



2

Oracle Text Indexing Elements

Oracle provides indexing types for storage, filtering, and lexers, and preferences and stoplists that you can use to create an Oracle Text index.

The chapter includes the following topics:

- Overview
- Creating Preferences
- Datastore Types
- Filter Types
- Lexer Types
- Wordlist Type
- Storage Types
- Section Group Types
- Classifier Types
- Cluster Types
- Stoplists
- System-Defined Preferences
- System Parameters
- Token Limitations for Oracle Text Indexes
- Auditing Oracle Text DR\$ Index Tables

2.1 Overview

When you use the CREATE INDEX statement to create an index or the ALTER INDEX statement to manage an index, you can optionally specify indexing preferences, stoplists, and section groups in the parameter string. Specifying a preference, stoplist, or section group answers one of the following questions about the way Oracle Text indexes text:

Preference Class	Answers the Question
Datastore	How are your documents stored?
Filter	How can the documents be converted to plain text?
Lexer	What language is being indexed?
Wordlist	How should stem and fuzzy queries be expanded?
Storage	How should the index tables be stored?
Stop List	What words or themes are not to be indexed?
Section Group	Is querying within sections enabled, and how are the document sections defined?

This chapter describes how to set each preference. Enable an option by creating a preference with one of the types described in this chapter.

2.2 Creating Preferences

To create a datastore, lexer, filter, classifier, wordlist, or storage preference, use the CTX_DDL.CREATE_PREFERENCE procedure and specify one of the types described in this chapter. For some types, you can also set attributes with the CTX_DDL.SET_ATTRIBUTE procedure.

An indexing *type* names a class of indexing objects that you can use to create an index *preference*. A type, therefore, is an abstract ID, while a preference is an entity that corresponds to a type. Many system-defined preferences have the same name as types (for example, BASIC_LEXER), but exact correspondence is not guaranteed. Be careful in assuming the existence or nature of either indexing types or system preferences.

You specify indexing preferences with the CREATE INDEX and ALTER INDEX statements. Indexing preferences determine how your index is created. For example, lexer preferences indicate the language of the text to be indexed. You can create and specify your own user-defined preferences, or you can use system-defined preferences.

To create a stoplist, use the CTX_DDL.CREATE_STOPLIST procedure. Add stopwords to a stoplist with CTX DDL.ADD STOPWORD.

To create section groups, use CTX_DDL.CREATE_SECTION_GROUP and specify a section group type. Add sections to section groups with the CTX_DDL.ADD_ZONE_SECTION or CTX_DDL.ADD_FIELD_SECTION procedures.

2.3 Datastore Types

Use the datastore types to create a datastore preference. This helps you specify how your text is stored.

Table 2-1 Datastore Types

Datastore Type	Use When
DIRECT_DATASTORE	Data is stored internally in the text column. Each row is indexed as a single document.
MULTI_COLUMN_DATASTORE	Data is stored in a text table in more than one column. Columns are concatenated to create a virtual document, one for each row.
DETAIL_DATASTORE	Data is stored internally in the text column. Document consists of one or more rows stored in a text column in a detail table, with header information stored in the primary table.



Table 2-1 (Cont.) Datastore Types

Datastore Type	Use When
FILE_DATASTORE	Data is stored externally in operating system files. File names are stored in the text column, one for each row.
	Note: Starting with Oracle Database 19c, the Oracle Text type FILE_DATASTORE is deprecated. Use DIRECTORY_DATASTORE instead.
DIRECTORY_DATASTORE	Data is stored in Oracle directory objects. File names are stored in the text column, one for each row.
NESTED_DATASTORE	Data is stored in a nested table.
URL_DATASTORE	Data is stored externally in files located on an intranet or the Internet. Uniform Resource Locators (URLs) are stored in the text column.
	Note: Starting with Oracle Database 19c, the Oracle Text type URL_DATASTORE is deprecated. Use NETWORK_DATASTORE instead.
NETWORK_DATASTORE	Data is stored externally in files located on an intranet or the Internet. Uniform Resource Locators (URLs) are stored in the text column.
USER_DATASTORE	Documents are synthesized at index time by a user- defined stored procedure.

2.3.1 DIRECT_DATASTORE

Use the $\protect_{DATASTORE}$ type for text stored directly in the text column, one document for each row. The $\protect_{DATASTORE}$ type has no attributes.

The following column types are supported: CHAR, VARCHAR, VARCHAR2, BLOB, CLOB, BFILE, XMLType, and URIType.



If your column is a ${\tt BFILE}$, then the index owner must have *read* permission on all directories used by the ${\tt BFILEs}$.

The following example creates a table with a CLOB column to store text data. It then populates two rows with text data and indexes the table using the system-defined preference CTXSYS.DEFAULT DATASTORE.

```
create table mytable(id number primary key, docs clob);
insert into mytable values(111555,'this text will be indexed');
insert into mytable values(111556,'this is a direct_datastore example');
commit;

create index myindex on mytable(docs)
  indextype is ctxsys.context
  parameters ('DATASTORE CTXSYS.DEFAULT DATASTORE');
```

2.3.2 MULTI_COLUMN_DATASTORE

Use the MULTI_COLUMN_DATASTORE datastore when your text is stored in more than one column. During indexing, the system concatenates the text columns, tags the column text, and indexes the text as a single document. The XML-like tagging is optional. You can also set the system to filter and concatenate binary columns.

- MULTI_COLUMN_DATASTORE Attributes
- Indexing and DML
- MULTI_COLUMN_DATASTORE Restriction
- MULTI_COLUMN_DATASTORE Example
- MULTI_COLUMN_DATASTORE Filter Example
- · Tagging Behavior
- Indexing Columns as Sections

2.3.2.1 MULTI COLUMN DATASTORE Attributes

The data store MULTI COLUMN DATASTORE has the attributes shown in Table 2-2.

Table 2-2 MULTI_COLUMN_DATASTORE Attributes

Attribute	Attribute Value
columns	Specify a comma-delimited list of columns to be concatenated during indexing. You can also specify any allowed expression for the SELECT statement column list for the base table. This includes expressions, PL/SQL functions, column aliases, and so on.
	The <code>NUMBER</code> and <code>DATE</code> column types are supported. They are converted to text before indexing using the default format mask. The <code>TO_CHAR</code> function can be used in the column list for formatting.
	The RAW and BLOB columns are directly concatenated as binary data.
	The LONG, LONG RAW, NCHAR, and NCLOB data types, nested table columns, and collections are not supported.
	The column list is limited to 500 bytes.
filter	Specify a comma-delimited list of Y/N flags. Each flag corresponds to a column in the COLUMNS list and denotes whether to filter the column using the AUTO_FILTER.
	Specify one of the following allowed values:
	Y: Column is to be filtered with AUTO_FILTER
	N or no value: Column is not to be filtered (default)
delimiter	Specify the delimiter that separates column text as follows:
	COLUMN_NAME_TAG: Column text is set off by XML-like open and close tags (default).
	NEWLINE: Column text is separated with a newline.

2.3.2.2 Indexing and DML

To index, you must create a dummy column to specify in the CREATE INDEX statement. This column's contents are not made part of the virtual document, unless its name is specified in the columns attribute.

The index is synchronized only when the dummy column is updated. You can create triggers to propagate changes if needed.

2.3.2.3 MULTI_COLUMN_DATASTORE Restriction

You cannot create a multicolumn datastore with <code>XMLType</code> columns. <code>MULTI_COLUMN_DATA_STORE</code> does not support <code>XMLType</code>. You can create a <code>CONTEXT</code> index with an <code>XMLType</code> column, as described in Oracle Text SQL Statements and Operators .

2.3.2.4 MULTI_COLUMN_DATASTORE Example

The following example creates a multicolumn datastore preference called my_multi with three text columns:

```
begin

ctx_ddl.create_preference('my_multi', 'MULTI_COLUMN_DATASTORE');
ctx_ddl.set_attribute('my_multi', 'columns', 'column1, column2, column3');
end;
```



2.3.2.5 MULTI_COLUMN_DATASTORE Filter Example

The following example creates a multicolumn datastore preference and denotes that the bar column is to be filtered with the AUTO FILTER.

```
ctx_ddl.create_preference('MY_MULTI','MULTI_COLUMN_DATASTORE');
ctx_ddl.set_attribute('MY_MULTI', 'COLUMNS','foo,bar');
ctx_ddl.set_attribute('MY_MULTI','FILTER','N,Y');
```

The multicolumn datastore fetches the content of the foo and bar columns, filters bar, then composes the compound document as:

```
<F00>
foo contents
</F00>
<BAR>
bar filtered contents (probably originally HTML)
</BAR>
```

The N flags do not need not be specified, and there does not need to be a flag for every column. Only the Y flags must be specified, with commas to denote which column they apply to. For example:

```
ctx_ddl.create_preference('MY_MULTI','MULTI_COLUMN_DATASTORE');
ctx_ddl.set_attribute('MY_MULTI', 'COLUMNS','foo,bar,zoo,jar');
ctx_ddl.set_attribute('MY_MULTI','FILTER',',,Y');
```

This example filters only the column zoo.

2.3.2.6 Tagging Behavior

During indexing, the system creates a virtual document for each row. The virtual document is composed of the contents of the columns concatenated in the listing order with column name tags automatically added.

For example:

```
create table mc(id number primary key, name varchar2(10), address varchar2(80));
insert into mc values(1, 'John Smith', '123 Main Street');
exec ctx_ddl.create_preference('mymds', 'MULTI_COLUMN_DATASTORE');
exec ctx_ddl.set_attibute('mymds', 'columns', 'name, address');
```

This produces the following virtual text for indexing:

```
<NAME>
John Smith
</NAME>
<ADDRESS>
123 Main Street
</ADDRESS>
```

2.3.2.7 Indexing Columns as Sections

To index tags as sections, you can optionally create field sections with BASIC SECTION GROUP.



No section group is created when you use the <code>MULTI_COLUMN_DATASTORE</code>. To create sections for these tags, you must create a section group.

When you use expressions or functions, the tag is composed of the first 30 characters of the expression unless a column alias is used.

For example, if your expression is as follows:

```
exec ctx ddl.set attibute('mymds', 'columns', '4 + 17');
```

then it produces the following virtual text:

```
<4 + 17>
21
</4 + 17>
```

If your expression is as follows:

```
exec ctx_ddl.set_attibute('mymds', 'columns', '4 + 17 col1');
```

then it produces the following virtual text:

```
<col1>
21
<col1>
```

The tags are in uppercase unless the column name or column alias is in lowercase and surrounded by double quotation marks. For example:

```
exec ctx ddl.set attibute('mymds', 'COLUMNS', 'foo');
```

This produces the following virtual text:

```
<FOO>
content of foo
</FOO>
```

For lowercase tags, use the following:

```
exec ctx ddl.set attibute('mymds', 'COLUMNS', 'foo "foo"');
```

This expression produces:

```
<foo>
content of foo
</foo>
```

2.3.3 DETAIL DATASTORE

Use the DETAIL_DATASTORE type for text stored directly in the database in detail tables, with the indexed text column located in the primary table.

- DETAIL_DATASTORE Attributes
- Synchronizing Primary/Detail Indexes
- Example Primary/Detail Tables



2.3.3.1 DETAIL DATASTORE Attributes

The DETAIL DATASTORE type has the attributes described in Table 2-3.

Table 2-3 DETAIL DATASTORE Attributes

Attribute	Attribute Value
binary	Specify TRUE for Oracle Text to add <i>no</i> newline character after each detail row.
	Specify FALSE for Oracle Text to add a newline character (\n) after each detail row automatically.
detail_table	Specify the name of the detail table (OWNER.TABLE if necessary).
detail_key	Specify the name of the detail table foreign key column.
detail_lineno	Specify the name of the detail table sequence column.
detail_text	Specify the name of the detail table text column.

2.3.3.2 Synchronizing Primary/Detail Indexes

Changes to the detail table do not trigger re-indexing when you synchronize the index. Only changes to the indexed column in the primary table triggers a re-index when you synchronize the index.

You can create triggers on the detail table to propagate changes to the indexed column in the primary table row.

2.3.3.3 Example Primary/Detail Tables

This example illustrates how primary and detail tables are related to each other.

- Primary Table Example
- Detail Table Example
- Detail Table Example Attributes
- Primary/Detail Index Example

2.3.3.3.1 Primary Table Example

Primary tables define the documents in a primary/detail relationship. Assign an identifying number to each document. The following table is an example primary table, called my primary:

Column Name	Column Type	Description
article_id	NUMBER	Document ID, unique for each document (primary key)
author	VARCHAR2(30)	Author of document
title	VARCHAR2(50)	Title of document
body	CHAR(1)	Dummy column to specify in CREATE INDEX





Your primary table must include a primary key column when you use the ${\tt DETAIL_DATASTORE}$ type.

2.3.3.3.2 Detail Table Example

Detail tables contain the text for a document, whose content is usually stored across a number of rows.

The following detail table my_{detail} is related to the primary table $my_{primary}$ with the $article_{id}$ column. This column identifies the primary document to which each detail row (sub-document) belongs.

Column Name	Column Type	Description
article_id	NUMBER	Document ID that relates to primary table
seq	NUMBER	Sequence of document in the primary document defined by article_id
text	VARCHAR2	Document text

2.3.3.3 Detail Table Example Attributes

In this example, the DETAIL DATASTORE attributes have the following values:

Attribute	Attribute Value
binary	TRUE
detail_table	my_detail
detail_key	article_id
detail_lineno	seq
detail_text	text

Use CTX_DDL.CREATE_PREFERENCE to create a preference with <code>DETAIL_DATASTORE</code>. Use CTX_DDL.SET_ATTRIBUTE to set the attributes for this preference as described earlier. The following example shows how this is done:

```
ctx_ddl.create_preference('my_detail_pref', 'DETAIL_DATASTORE');
ctx_ddl.set_attribute('my_detail_pref', 'binary', 'true');
ctx_ddl.set_attribute('my_detail_pref', 'detail_table', 'my_detail');
ctx_ddl.set_attribute('my_detail_pref', 'detail_key', 'article_id');
ctx_ddl.set_attribute('my_detail_pref', 'detail_lineno', 'seq');
ctx_ddl.set_attribute('my_detail_pref', 'detail_text', 'text');
end;
```

2.3.3.3.4 Primary/Detail Index Example

To index the document defined in this primary/detail relationship, specify a column in the primary table using the CREATE INDEX statement.

The column you specify must be one of the allowed types.

This example uses the <code>body</code> column, whose function is to enable the creation of the primary/ detail index and to improve readability of the code. The <code>my_detail_pref</code> preference is set to <code>DETAIL_DATASTORE</code> with the required attributes:

```
CREATE INDEX myindex on my_primary(body) indextype is ctxsys.context parameters('datastore my detail pref');
```

In this example, you can also specify the title or author column to create the index. However, if you do so, changes to these columns will trigger a re-index operation.

2.3.4 FILE_DATASTORE

The FILE DATASTORE type is used for text stored in files accessed through the local file system.



Starting with Oracle Database 19c, the Oracle Text type <code>FILE_DATASTORE</code> is deprecated. Use <code>DIRECTORY_DATASTORE</code> instead.

Oracle recommends that you replace <code>FILE_DATASTORE</code> text indexes with the <code>DIRECTORY_DATASTORE</code> index type, which is available starting with Oracle Database <code>19c. DIRECTORY_DATASTORE</code> provides greater security because it enables file access to be based on directory objects.

Note:

- The FILE_DATASTORE type may not work with certain types of remote-mounted file systems.
- The character set of the file datastore is assumed to be the character set of the database.
- FILE_DATASTORE Attributes
- FILE DATASTORE and Security
- FILE_DATASTORE Example

2.3.4.1 FILE DATASTORE Attributes

The FILE DATASTORE type has the attributes described Table 2-4.

Table 2-4 FILE_DATASTORE Attributes

Attribute	Attribute Value
path	path1:path2:pathn
filename_charset	name



path

Specifies the full directory path name of the files stored externally in a file system. When you specify the full directory path as such, you need to include only file names in your text column. You can specify multiple paths for the path attribute, with each path separated by a colon (:) on UNIX and semicolon(;) on Windows. File names are stored in the text column in the text table.

If you do not specify a path for external files with this attribute, then Oracle Text requires that the path be included in the file names stored in the text column.

The PATH attribute has the following limitations:

- If you specify a PATH attribute, then you can only use a simple file name in the indexed column. You cannot combine the PATH attribute with a path as part of the file name. If the files exist in multiple folders or directories, you must leave the PATH attribute unset, and include the full file name, with PATH, in the indexed column.
- On Windows systems, the files must be located on a local drive. They cannot be on a remote drive, whether the remote drive is mapped to a local drive letter.

filename_charset

Specifies a valid Oracle character set name (maximum length 30 characters) to be used by the file datastore for converting file names. In general, the Oracle database can use a different character set than the operating system. This can lead to problems in finding files (which may raise DRG-11513 errors) when the indexed column contains characters that are not convertible to the operating system character set. By default, the file datastore will convert the file name to WE8ISO8859p1 for ASCII platforms or WE8EBCDIC1047 for EBCDIC platforms. However, this may not be sufficient for applications with multibyte character sets for both the database and the operating system, because neither WE8ISO8859p1 nor WE8EBCDIC1047 supports multibyte characters. The attribute filename_charset rectifies this problem. If specified, then the datastore will convert from the database character set to the specified character set rather than to ISO8859 or EBCDIC.

If the filename_charset attribute is the same as the database character set, then the file name is used as is. If filename_charset is not a valid character set, then the error "DRG-10763: value %s is not a valid character set" is raised.

2.3.4.2 FILE_DATASTORE and Security

File and URL datastores enable access to files on the actual database disk. This may be undesirable when security is an issue since any user can browse the file system that is accessible to the Oracle user. Any user attempting to create an index using <code>FILE</code> or <code>URL</code> datastores must have the <code>TEXT DATASTORE ACCESS</code> system privilege granted to the user directly, or the index creation will fail. Granting the user <code>TEXT DATASTORE ACCESS</code> privilege indirectly by granting it to the user's role does not work and the index creation will still fail. Thus, by default, users are not able to create indexes that use the <code>FILE</code> or <code>URL</code> datastores. Granting <code>TEXT DATASTORE ACCESS</code> to <code>PUBLIC</code> gives any user the privilege to index either an arbitrary file in the file system in the case of <code>FILE</code> datastore and an arbitrary URL in the case of <code>URL</code> datastore and is not recommended.

For example, the following statement grants TEXT DATASTORE ACCESS to the user SCOTT:

grant TEXT DATASTORE ACCESS to SCOTT;

The CREATE INDEX operation will fail when a user that does not have TEXT DATASTORE ACCESS privilege tries to create an index on a FILE or URL datastore. For example:

CREATE INDEX myindex ON mydocument(TEXT) INDEXTYPE IS ctxsys.context PARAMETERS('DATASTORE ctxsys.file_datastore')



In this case, if the user does not have the TEXT DATASTORE ACCESS privilege granted directly to it, then index creation will fail and returns an error. For users who have the TEXT DATASTORE ACCESS privilege, the index creation will proceed normally.

The user's privilege is checked any time the datastore is accessed. This includes index creation, index sync, and calls to document services, such as CTX_DOC.HIGHLIGHT.

2.3.4.3 FILE_DATASTORE Example

This example creates a file datastore preference called COMMON_DIR that has a path of /mydocs:

```
begin
  ctx_ddl.create_preference('COMMON_DIR','FILE_DATASTORE');
  ctx_ddl.set_attribute('COMMON_DIR','PATH','/mydocs');
end;
```

When you populate the table mytable, you need only insert file names. The path attribute tells the system where to look during the indexing operation.

```
create table mytable(id number primary key, docs varchar2(2000));
insert into mytable values(111555, 'first.txt');
insert into mytable values(111556, 'second.txt');
commit;
```

Create the index as follows:

```
create index myindex on mytable(docs)
indextype is ctxsys.context
parameters ('datastore COMMON DIR');
```

2.3.5 DIRECTORY DATASTORE

Use the DIRECTORY_DATASTORE type during indexing to access the text stored in files which can be accessed through Oracle directory objects.

Starting with Oracle Database 19c, the Oracle Text type <code>FILE_DATASTORE</code> is deprecated. Use <code>DIRECTORY DATASTORE</code> instead.

Oracle recommends that you replace <code>FILE_DATASTORE</code> text indexes with the <code>DIRECTORY_DATASTORE</code> index type, which is available starting with Oracle Database 19c. <code>DIRECTORY_DATASTORE</code> provides greater security because it enables file access to be based on directory objects.

A directory object specifies an alias for a directory on the server file system where external binary file LOBs (BFILES) and external table data are located. When you use DIRECTORY_DATASTORE type, another PDB user can not access directory objects in your PDB without read access to the directory objects.

Use the <code>DIRECTORY_DATASTORE</code> type to use an Oracle directory object as an attribute for the <code>CTX_DDL.SET_ATTRIBUTE</code> procedure. You must have read access to the Oracle directory object to access the files stored within the directory. If you have access, then during index creation, you can use the path stored in the Oracle directory object to access the files stored in the file system.

- DIRECTORY_DATASTORE Attributes
- DIRECTORY_DATASTORE Example



Note:

- To create an Oracle directory object, you must have the CREATE ANY DIRECTORY
 privilege. Typically, a system administrator user creates the directory and
 provides read access to the directory for an Oracle Text user.
- DIRECTORY_DATASTORE can be used with a context index on CHAR datatype column only if the file name fills the column.

2.3.5.1 DIRECTORY_DATASTORE Attributes

DIRECTORY DATASTORE has the following attributes:

Table 2-5 DIRECTORY_DATASTORE Attributes

Attribute	Attribute Values
directory	Specify the name of the directory object where the data to be indexed is stored. The default is ${\tt NULL}$.
	If you have access to the Oracle directory object, then you can also access the files in its sub-directories.
filename_charset	Specify a valid Oracle character set name (maximum length 30 characters) to be used by the directory datastore for converting file names.
	In general, the Oracle database can use a different character set than the operating system. This can lead to problems in finding files (which may raise DRG-11513 errors) when the indexed column contains characters that are not convertible to the operating system character set. By default, the directory datastore will convert the file name to WE8ISO8859p1 for ASCII platforms or WE8EBCDIC1047 for EBCDIC platforms.
	However, this may not be sufficient for applications with multibyte character sets for both the database and the operating system, because neither WE8ISO8859p1 nor WE8EBCDIC1047 supports multibyte characters. The attribute filename_charset rectifies this problem. If specified, then the datastore will convert from the database character set to the specified character set rather than to ISO8859 or EBCDIC.
	If the filename_charset attribute is the same as the database character set, then the file name is used as is. If filename_charset is not a valid character set, then the error "DRG-10763: value %s is not a valid character set" is raised.

2.3.5.2 DIRECTORY_DATASTORE Example

This example shows you how to create an index with <code>DIRECTORY_DATASTORE</code> type by securely accessing files under an Oracle directory object.

Create an Oracle directory object to store the path of the files. You must have the CREATE ANY DIRECTORY privilege to create an Oracle directory object.

create directory myhome as 'directory path';



Create a directory datastore preference called MYDS and set the directory attribute with myhome, which is the Oracle directory object:

```
exec ctx_ddl.create_preference('MYDS','DIRECTORY_DATASTORE')
exec ctx ddl.set attribute('MYDS','DIRECTORY','myhome')
```

Create a table named mytable and populate it with file names only. The directory attribute tells the system where to look during the indexing operation.

```
create table mytable(id number primary key, docs varchar2(2000));
insert into mytable values(111555,'first.txt');
insert into mytable values(111556,'second.txt');
```

Create the index as follows:

```
create index myindex on mytable(docs)
indextype is ctxsys.context
parameters ('datastore MYDS');
```

2.3.6 URL DATASTORE

Use the URL_DATASTORE type for text stored in files on the World Wide Web (accessed through HTTP or FTP) and local file system (accessed through the file protocol).

Store each URL in a single text field.

Note:

Starting with Oracle Database 19c, the Oracle Text type <code>URL_DATASTORE</code> is deprecated. Use <code>NETWORK_DATASTORE</code> instead.

The URL_DATASTORE type is used for text stored in files on the internet (accessed through HTTP or FTP), and for text stored in local file system files (accessed through the file protocol). It is replaced with NETWORK_DATASTORE, which uses ACLs to allow access to specific servers. This change aligns Oracle Text more closely with the standard operating and security model for accessing URLs from the database.

- URL_DATASTORE URL Syntax
- URL_DATASTORE Attributes
- URL DATASTORE and Security
- URL DATASTORE Example

2.3.6.1 URL_DATASTORE URL Syntax

The syntax of a URL you store in a text field is as follows (with brackets indicating optional parameters):

```
[URL:]<access_scheme>://<host_name>[:<port_number>]/[<url_path>]
```

The access scheme string can be either ftp, http, or file. For example:



http://mycomputer.us.example.com/home.html



The login:password@ syntax within the URL is supported only for the ftp access scheme.

Because this syntax is partially compliant with the RFC 1738 specification, the following restriction holds for the URL syntax: The URL must contain only printable ASCII characters. Non-printable ASCII characters and multibyte characters must be escaped with the %xx notation, where xx is the hexadecimal representation of the special character.

2.3.6.2 URL_DATASTORE Attributes

URL DATASTORE has the following attributes:

Table 2-6 URL_DATASTORE Attributes

Attribute	Attribute Value
timeout	The value of this attribute is ignored. This is provided for backward compatibility.
maxthreads	The value of this attribute is ignored. URL_DATASTORE is single-threaded. This is provided for backward compatibility.
urlsize	The value of this attribute is ignored. This is provided for backward compatibility.
maxurls	The value of this attribute is ignored. This is provided for backward compatibility.
maxdocsize	The value of this attribute is ignored. This is provided for backward compatibility.
http_proxy	Specify the host name of http proxy server. Optionally specify port number with a colon in the form hostname:port.
ftp_proxy	Specify the host name of ftp proxy server. Optionally specify port number with a colon in the form hostname:port.
no_proxy	Specify the domain for no proxy server. Use a comma-delimited string of up to 16 domain names.

timeout

The value of this attribute is ignored. This is provided for backward compatibility.

maxthreads

The value of this attribute is ignored. URL_DATASTORE is single-threaded. This is provided for backward compatibility.

urlsize

The value of this attribute is ignored. This is provided for backward compatibility.

maxdocsize

The value of this attribute is ignored. This is provided for backward compatibility.



maxurls

The value of this attribute is ignored. This is provided for backward compatibility.

http_proxy

Specify the fully qualified name of the host computer that serves as the HTTP proxy (gateway) for the computer on which Oracle Text is installed. You can optionally specify port number with a colon in the form hostname:port.

You must set this attribute if the computer is in an intranet that requires authentication through a proxy server to access Web files located outside the firewall.

ftp_proxy

Specify the fully qualified name of the host computer that serves as the FTP proxy (gateway) for the server on which Oracle Text is installed. You can optionally specify a port number with a colon in the form hostname:port.

This attribute must be set if the computer is in an intranet that requires authentication through a proxy server to access Web files located outside the firewall.

no_proxy

Specify a string of domains (up to sixteen, separated by commas) that are found in most, if not all, of the computers in your intranet. When one of the domains is encountered in a host name, no request is sent to the server(s) specified for ftp_proxy and http_proxy. Instead, the request is processed directly by the host computer identified in the URL.

For example, if the string *us.example.com*, *uk.example.com* is entered for no_proxy, any URL requests to computers that contain either of these domains in their host names are not processed by your proxy server(s).

2.3.6.3 URL DATASTORE and Security

For a discussion of how to control file access security for file and URL datastores, refer to "FILE DATASTORE and Security".

2.3.6.4 URL DATASTORE Example

This example creates a <code>URL_DATASTORE</code> preference called <code>URL_PREF</code> for which the <code>http_proxy</code>, <code>no_proxy</code>, and <code>timeout</code> attributes are set. The defaults are used for the attributes that are not set.

```
begin
  ctx_ddl.create_preference('URL_PREF','URL_DATASTORE');
  ctx_ddl.set_attribute('URL_PREF','HTTP_PROXY','www-proxy.us.example.com');
  ctx_ddl.set_attribute('URL_PREF','NO_PROXY','us.example.com');
  ctx_ddl.set_attribute('URL_PREF','Timeout','300');
end:
```

Create the table and insert values into it:

```
create table urls(id number primary key, docs varchar2(2000));
insert into urls values(111555,'http://context.us.example.com');
insert into urls values(111556,'http://www.sun.com');
commit;
```

To create the index, specify URL PREF as the datastore:

```
create index datastores_text on urls ( docs )
  indextype is ctxsys.context
  parameters ( 'Datastore URL_PREF' );
```



2.3.7 NETWORK_DATASTORE

Use the NETWORK_DATASTORE type during indexing to access the files stored on the World Wide Web through HTTP and HTTPS.

Starting with Oracle Database 19c, the Oracle Text type <code>URL_DATASTORE</code> is deprecated. Use <code>NETWORK DATASTORE</code> instead.

The URL_DATASTORE type is used for text stored in files on the internet (accessed through HTTP or FTP), and for text stored in local file system files (accessed through the file protocol). It is replaced with NETWORK_DATASTORE, which uses ACLs to allow access to specific servers. This change aligns Oracle Text more closely with the standard operating and security model for accessing URLs from the database.

When you use NETWORK_DATASTORE type, you can access a URL after the website certificate is verified in Oracle wallet and ACL package.

FTP and file protocol are not supported in <code>NETWORK_DATASTORE</code> type. To access the files stored in the local file system, use the <code>DIRECTORY DATASTORE</code> type.

During index creation, the URL stored in the datastore is used to access the files stored in the World Wide Web. The access is granted after verifying the website certificate in Oracle wallet.

- NETWORK_DATASTORE URL Syntax
- NETWORK DATASTORE Attributes
- NETWORK_DATASTORE Example



NETWORK_DATASTORE can be used with a context index on CHAR datatype column only if the file name fills the column.

2.3.7.1 NETWORK_DATASTORE URL Syntax

The syntax of a URL you store in a datastore is as follows (with brackets indicating optional parameters):

```
[URL:] <access scheme>://<host name>[:<port number>]/[<url path>]
```

The access scheme string can be either http or https. For example:

https://mycomputer.us.example.com/home.html

Because this syntax is partially compliant with the RFC 1738 specification, the following restriction holds for the URL syntax: The URL must contain only printable ASCII characters. Non-printable ASCII characters and multibyte characters must be escaped with the %xx notation, where xx is the hexadecimal representation of the special character.



2.3.7.2 NETWORK_DATASTORE Attributes

Use these attributes with the $\verb|NETWORK_DATASTORE|$ type during indexing, for text stored in files on the internet or in local file system files.

Table 2-7 NETWORK_DATASTORE Attributes

Attribute	Attribute Value
timeout	Specify the time out value for all future HTTP requests that use the UTL_HTTP package to read the HTTP response from a web or proxy server. This attribute can be used to avoid being blocked by busy web servers or heavy network traffic when retrieving web pages.
	The default value is 30 seconds. The minimum value is 1 second and the maximum value is 3600 seconds.
http_proxy	Specify the fully qualified name of the host computer that serves as the HTTP proxy (gateway) for the computer on which Oracle Text is installed. You can optionally specify port number with a colon in the form hostname:port.
	You must set this attribute if the computer is in an intranet that requires authentication through a proxy server to access Web files located outside the firewall.
	For HTTP network connection, an ACL package is required so that the UTL_HTTP package can interact with the external host. You must have EXECUTE privilege for the DBMS_NETWORK_ACL_ADMIN package to grant the CONNECT privilege on the ACL to a user.
https_proxy	Specify the fully qualified name of the host computer that serves as the HTTPS proxy (gateway) for the computer on which Oracle Text is installed. You can optionally specify port number with a colon in the form hostname:port.
	You must set this attribute if the computer is in an intranet that requires authentication through a proxy server to access Web files located outside the firewall.
	For HTTPS network connection, in addition to the ACL package, an Oracle wallet is also required. You can create an Oracle wallet using the orapki command-line utility.
	To create an Oracle wallet using the <code>orapki</code> command-line utility, use the <code>orapki</code> wallet <code>create</code> command:
	<pre>orapki wallet create -wallet wallet_location -pwd password -auto_login</pre>
	To add a trusted certificate to an Oracle wallet, use the orapki wallet add command:
	<pre>orapki wallet add -wallet wallet_location -trusted_cert -cert certificate_location -pwd password</pre>
	Use the <code>UTL_HTTP.SET_WALLET</code> procedure to configure the request to hold the wallet:
	<pre>EXEC UTL_HTTP.SET_WALLET(wallet_location, password);</pre>

Table 2-7 (Cont.) NETWORK_DATASTORE Attributes

Attribute	Attribute Value
no_proxy	Specify a string of domains (up to sixteen, separated by commas) that are found in most, if not all, of the computers in your intranet. When one of the domains is encountered in a host name, no request is sent to the server(s) specified for http_proxy and https_proxy. Instead, the request is processed directly by the host computer identified in the URL.
	For example, if the string us.example.com, uk.example.com is entered for no_proxy, any URL requests to computers that contain either of these domains in their host names are not processed by your proxy server(s).

Related Topics

- DBMS_NETWORK_ACL_ADMIN
- UTL_HTTP

2.3.7.3 NETWORK_DATASTORE Example

This example shows you how to configure HTTP and HTTPS network connections and create an index based on the <code>NETWORK_DATASTORE</code> type to access the files stored on the World Wide Web.

Create a user and grant the necessary privileges:

```
CREATE USER myuser IDENTIFIED by password;
GRANT connect, resource, unlimited tablespace, ctxapp to myuser;
```

Append an access control entry (ACE) to the ACL of a network host. The ACL controls access to the given host from the database and the ACE specifies the privileges granted to or denied from the specified principal. When host is specified as '*', you can access any host through the network datastore which uses UTL_HTTP package internally to access data from websites through HTTP.

Create a network datastore preference called NETWORK_PREF:

```
begin
  ctx_ddl.create_preference('NETWORK_PREF','NETWORK_DATASTORE');
  ctx_ddl.set_attribute('NETWORK_PREF','HTTP_PROXY','www-
proxy.us.example.com');
  ctx_ddl.set_attribute('NETWORK_PREF','NO_PROXY','us.example.com');
  ctx_ddl.set_attribute('NETWORK_PREF','TIMEOUT','300');
```



```
end;
```

Create a table named mytable and populate it with URLs:

```
create table mytable(id number primary key, docs varchar2(2000));
insert into mytable values(111555, 'http://context.example.com');
insert into mytable values(111556, 'http://www.johndoe.com');
```

Create the index as follows:

```
create index myindex on mytable(docs)
  indextype is ctxsys.context
  parameters ('datastore NETWORK PREF');
```

See Also:

- Oracle Database PL/SQL Packages and Types Reference for more information about DBMS NETWORK ACL ADMIN package
- Oracle Database PL/SQL Packages and Types Reference for more information about UTL HTTP package

2.3.8 USER_DATASTORE

Use the USER_DATASTORE type to define stored procedures that synthesize documents during indexing. For example, a user procedure might synthesize author, date, and text columns into one document to have the author and date information be part of the indexed text.

- USER_DATASTORE Attributes
- USER DATASTORE Constraints
- USER_DATASTORE Editing Procedure after Indexing
- USER_DATASTORE with CLOB Example
- USER_DATASTORE with BLOB_LOC Example

2.3.8.1 USER DATASTORE Attributes

USER DATASTORE has the following attributes:

Table 2-8 USER_DATASTORE Attributes

Attribute	Attribute Value
procedure	Specify the procedure that synthesizes the document to be indexed.
	This procedure can be owned by any user and must be executable by the index
	owner.



Table 2-8 (Cont.) USER_DATASTORE Attributes

Attribute	Attribute Value
output_type	Specify the data type of the second argument to procedure. Valid values are CLOB, BLOB, CLOB_LOC, BLOB_LOC, or VARCHAR2. The default is CLOB.
	When you specify CLOB_LOC, BLOB_LOC, you indicate that no temporary CLOB or BLOB is needed, because your procedure copies a locator to the IN/OUT second parameter.

procedure

Specify the name of the procedure that synthesizes the document to be indexed. This specification must be in the form $\protect{PROCEDURENAME}$ or $\protect{PACKAGENAME}$. $\protect{PROCEDURENAME}$. You can also specify the schema owner name.

The procedure you specify must have two arguments defined as follows:

procedure (r IN ROWID, c IN OUT NOCOPY output type)

The first argument r must be of type ROWID. The second argument c must be of the type specified in the <code>output_type</code> attribute. NOCOPY is a compiler hint that instructs Oracle Text to pass parameter c by reference if possible.



Procedure names should not include the semicolon character.

The stored procedure is called once for each row indexed. Given the rowid of the current row, procedure must write the text of the document into its second argument, whose type you specify with output type.

2.3.8.2 USER DATASTORE Constraints

The following constraints apply to procedure:

- It can be owned by any user, but the user must have database permissions to execute procedure correctly
- It must be executable by the index owner
- It must not enter DDL or transaction control statements, like COMMIT

2.3.8.3 USER DATASTORE Editing Procedure after Indexing

When you change or edit the stored procedure, indexes based on it will not be notified, so you must manually re-create such indexes. So if the stored procedure makes use of other columns, and those column values change, the row will not be re-indexed. The row is re-indexed only when the indexed column changes.

output type

Specify the datatype of the second argument to procedure. You can use either CLOB, BLOB, CLOB_LOC, BLOB_LOC, or VARCHAR2.



2.3.8.4 USER DATASTORE with CLOB Example

Consider a table in which the author, title, and text fields are separate, as in the articles table defined as follows:

```
create table articles(
   id     number,
   author   varchar2(80),
   title   varchar2(120),
   text   clob);
```

The author and title fields are to be part of the indexed document text. Assume user appowner writes a stored procedure with the user datastore interface that synthesizes a document from the text, author, and title fields:

This procedure takes in a rowid and a temporary CLOB locator, and concatenates all the article's columns into the temporary CLOB. The for loop executes only once.

The user appowner creates the preference as follows:

```
begin

ctx_ddl.create_preference('myud', 'user_datastore');
ctx_ddl.set_attribute('myud', 'procedure', 'myproc');
ctx_ddl.set_attribute('myud', 'output_type', 'CLOB');
end;
```

When appowner creates the index on articles (text) using this preference, the indexing operation sees author and title in the document text.

2.3.8.5 USER_DATASTORE with BLOB_LOC Example

The following procedure might be used with OUTPUT TYPE BLOB LOC:

```
procedure myds(rid in rowid, dataout in out nocopy blob)
is
    l_dtype varchar2(10);
    l_pk number;
begin
    select dtype, pk into l_dtype, l_pk from mytable where rowid = rid;
    if (l_dtype = 'MOVIE') then
        select movie_data into dataout from movietab where fk = l_pk;
    elsif (l_dtype = 'SOUND') then
        select sound_data into dataout from soundtab where fk = l_pk;
```



```
end if;
end;
```

The user appowner creates the preference as follows:

```
begin

ctx_ddl.create_preference('myud', 'user_datastore');
ctx_ddl.set_attribute('myud', 'procedure', 'myproc');
ctx_ddl.set_attribute('myud', 'output_type', 'blob_loc');
end;
```

2.3.9 NESTED DATASTORE

Use the nested datastore type to index documents stored as rows in a nested table.

- NESTED_DATASTORE Attributes
- NESTED_DATASTORE Example

2.3.9.1 NESTED DATASTORE Attributes

NESTED DATASTORE has the following attributes:

Table 2-9 NESTED DATASTORE Attributes

Attribute	Attribute Value
nested_column	Specify the name of the nested table column. This attribute is required. Specify only the column name. Do not specify schema owner or containing table name.
nested_type	Specify the type of nested table. This attribute is required. You must provide owner name and type.
nested_lineno	Specify the name of the attribute in the nested table that orders the lines. This is like <code>DETAIL_LINENO</code> in detail datastore. This attribute is required.
nested_text	Specify the name of the column in the nested table type that contains the text of the line. This is like <code>DETAIL_TEXT</code> in detail datastore. This attribute is required. <code>LONG</code> column types are not supported as nested table text columns.
binary	Specify FALSE for Oracle Text to automatically insert a newline character when synthesizing the document text. If you specify TRUE, Oracle Text does not do this. This attribute is not required. The default is FALSE.

When using the nested table datastore, you must index a dummy column, because the extensible indexing framework disallows indexing the nested table column. See "NESTED_DATASTORE Example".

DML on the nested table is not automatically propagated to the dummy column used for indexing. For DML on the nested table to be propagated to the dummy column, your application code or trigger must explicitly update the dummy column.

Filter defaults for the index are based on the type of the nested text column.

During validation, Oracle Text checks that the type exists and that the attributes you specify for nested_lineno and nested_text exist in the nested table type. Oracle Text does not check that the named nested table column exists in the indexed table.

2.3.9.2 NESTED DATASTORE Example

This section shows an example of using the <code>NESTED_DATASTORE</code> type to index documents stored as rows in a nested table.

- Create the Nested Table
- Insert Values into Nested Table
- Create Nested Table Preferences
- Create Index on Nested Table
- Query Nested Datastore

2.3.9.2.1 Create the Nested Table

The following code creates a nested table and a storage table mytab for the nested table:

```
create type nt_rec as object (
    lno number, -- line number
    ltxt varchar2(80) -- text of line
);

create type nt_tab as table of nt_rec;
create table mytab (
    id number primary key, -- primary key
    dummy char(1), -- dummy column for indexing
    doc nt_tab -- nested table
)
nested table doc store as myntab;
```

2.3.9.2.2 Insert Values into Nested Table

The following code inserts values into the nested table for the parent row with ID equal to 1.

```
insert into mytab values (1, null, nt_tab());
insert into table(select doc from mytab where id=1) values (1, 'the dog');
insert into table(select doc from mytab where id=1) values (2, 'sat on mat ');
commit;
```

2.3.9.2.3 Create Nested Table Preferences

The following code sets the preferences and attributes for the <code>NESTED_DATASTORE</code> according to the definitions of the nested table type <code>nt_tab</code> and the parent table <code>mytab</code>:

```
begin
-- create nested datastore pref
ctx_ddl.create_preference('ntds','nested_datastore');
-- nest tab column in main table
ctx_ddl.set_attribute('ntds','nested_column', 'doc');
-- nested table type
ctx_ddl.set_attribute('ntds','nested_type', 'scott.nt_tab');
-- lineno column in nested table
ctx_ddl.set_attribute('ntds','nested_lineno','lno');
--text column in nested table
```



```
ctx_ddl.set_attribute('ntds','nested_text', 'ltxt');
end;
```

2.3.9.2.4 Create Index on Nested Table

The following code creates the index using the nested table datastore:

```
create index myidx on mytab(dummy) -- index dummy column, not nest table
indextype is ctxsys.context parameters ('datastore ntds');
```

2.3.9.2.5 Query Nested Datastore

The following select statement queries the index built from a nested table:

```
select * from mytab where contains(dummy, 'dog and mat')>0;
-- returns document 1, because it has dog in line 1 and mat in line 2.
```

2.4 Filter Types

Use the filter types to create preferences that determine how text is filtered for indexing. Filters enable word processor documents, formatted documents, plain text, HTML, and XML documents to be indexed.

For formatted documents, Oracle Text stores documents in their native format and uses filters to build interim plain text or HTML versions of the documents. Oracle Text indexes the words derived from the plain text or HTML version of the formatted document.

To create a filter preference, you must use one of the filter types shown in Table 2-10.

Table 2-10 Filter Types

Filter	When Used
AUTO_FILTER	Auto filter for filtering formatted documents.
NULL_FILTER	No filtering required. Use for indexing plain text, HTML, or XML documents.
MAIL_FILTER	Use the ${\tt MAIL_FILTER}$ to transform RFC-822, RFC-2045 messages in to text that can be indexed.
USER_FILTER	User-defined external filter to be used for custom filtering.
PROCEDURE_FILTER	User-defined stored procedure filter to be used for custom filtering.

2.4.1 AUTO_FILTER

The AUTO_FILTER is a universal filter that filters most document formats, including PDF and Microsoft Word documents. Use it for indexing both single-format and mixed-format columns. This filter automatically bypasses plain text, HTML, XHTML, SGML, and XML documents.

- AUTO_FILTER Attributes
- AUTO FILTER and Indexing Formatted Documents
- AUTO FILTER and Explicitly Bypassing Plain Text or HTML in Mixed Format Columns
- AUTO FILTER and Character Set Conversion With AUTO FILTER



See Also:

Oracle Text Supported Document Formats, for a list of the formats supported by AUTO_FILTER, and to learn more about how to set up your environment

Note:

The AUTO_FILTER replaces the INSO_FILTER, which has been deprecated. While every effort has been made to ensure maximal backward compatibility between the two filters, so that applications using INSO_FILTER will continue to work without modification, some differences may arise. Users should therefore use AUTO_FILTER in their new programs and, when possible, replace instances of INSO_FILTER, and any system preferences or constants that make use of it, in older applications.

2.4.1.1 AUTO FILTER Attributes

The AUTO FILTER preference has the attributes shown in Table 2-11.

Table 2-11 AUTO FILTER Attributes

Attribute	Attribute Value
timeout	Specify the AUTO_FILTER timeout in seconds. Use a number between 0 and 42,949,672. Default is 120. Setting this value to 0 disables the feature.
	How this wait period is used depends on how you set timeout_type.
	This feature is disabled for rows for which the corresponding charset and format column cause the AUTO_FILTER to bypass the row, such as when format is marked TEXT.
	Use this feature to prevent the Oracle Text indexing operation from waiting indefinitely on a hanging filter operation.
timeout_type	Specify either HEURISTIC or FIXED. Default is HEURISTIC.
	Specify HEURISTIC for Oracle Text to check every TIMEOUT seconds if output from Outside In HTML Export has increased. The operation terminates for the document if output has not increased. An error is recorded in the CTX_USER_INDEX_ERRORS view and Oracle Text moves to the next document row to be indexed.
	Specify FIXED to terminate the Outside In HTML Export processing after TIMEOUT seconds regardless of whether filtering was progressing normally or just hanging. This value is useful when indexing throughput is more important than taking the time to successfully filter large documents.
output_formatting	Setting this attribute has no effect on filter performance or filter output. It is maintained for backward compatibility.

2.4.1.2 AUTO_FILTER and Indexing Formatted Documents

Use AUTO_FILTER to index a text column containing formatted documents, such as Microsoft Word. This filter automatically detects the document format.

Use the CTXSYS.AUTO FILTER system-defined preference in the parameter clause as follows:

```
create index hdocsx on hdocs(text) indextype is ctxsys.context
parameters ('datastore ctxsys.directory_datastore
filter ctxsys.auto_filter');
```



The CTXSYS.AUTO_FILTER replaces CTXSYS.INSO_FILTER, which has been deprecated. Programs making use of CTXSYS.INSO_FILTER should still work. New programs should use CTXSYS.AUTO FILTER.

2.4.1.3 AUTO_FILTER and Explicitly Bypassing Plain Text or HTML in Mixed Format Columns

The AUTO_FILTER can index mixed-format columns, automatically bypassing plain text, HTML, and XML documents. However, if you prefer not to depend on the built-in bypass mechanism, you can explicitly tag your rows as text and cause the AUTO_FILTER to ignore the row and not process the document in any way.

A mixed-format column is a text column containing more than one document format, such as a column that contains Microsoft Word, PDF, plain text, and HTML documents.

The format column in the base table enables you to specify the type of document contained in the text column. You can specify the following document types: <code>TEXT</code>, <code>BINARY</code>, and <code>IGNORE</code>. During indexing, the <code>AUTO_FILTER</code> ignores any document typed <code>TEXT</code>, assuming the charset column is not specified. The difference between a document with a <code>TEXT</code> format column type and one with an <code>IGNORE</code> type is that the <code>TEXT</code> document is indexed, but ignored by the filter, while the <code>IGNORE</code> document is not indexed at all. Use <code>IGNORE</code> to overlook documents such as image files, or documents in a language that you do not want to index. <code>IGNORE</code> can be used with any filter type.

To set up the AUTO_FILTER bypass mechanism, you must create a format column in your base table.

For example:

```
create table hdocs (
    id number primary key,
    fmt varchar2(10),
    text varchar2(80)
);
```

Assuming you are indexing mostly Word documents, you specify BINARY in the format column to filter the Word documents. Alternatively, to have the AUTO_FILTER ignore an HTML document, specify TEXT in the format column.

For example, the following statements add two documents to the text table, assigning one format as BINARY and the other TEXT:

```
insert into hdocs values(1, 'binary', '/docs/myword.doc');
insert in hdocs values (2, 'text', '/docs/index.html');
commit;
```

To create the index, use CREATE INDEX and specify the format column name in the parameter string:

```
create index hdocsx on hdocs(text) indextype is ctxsys.context
  parameters ('datastore ctxsys.directory_datastore
  filter ctxsys.auto_filter
  format column fmt');
```

If you do not specify TEXT or BINARY for the format column, BINARY is used.



You need not specify the format column in ${\tt CREATE}$ ${\tt INDEX}$ when using the ${\tt AUTO_FILTER}.$

2.4.1.4 AUTO_FILTER and Character Set Conversion With AUTO_FILTER

The AUTO_FILTER converts documents to the database character set when the document format column is set to TEXT. In this case, the AUTO_FILTER looks at the charset column to determine the document character set.

If the charset column value is not an Oracle Text character set name, the document is passed through without any character set conversion.



You need not specify the charset column when using the AUTO_FILTER.

2.4.2 NULL FILTER

Use the $\mathtt{NULL_FILTER}$ type when plain text or HTML is to be indexed and no filtering needs to be performed. \mathtt{NULL} FILTER has no attributes.

NULL FILTER and Indexing HTML Documents

If your document set is entirely HTML, Oracle recommends that you use the <code>NULL_FILTER</code> in your filter preference.

For example, to index an HTML document set, specify the system-defined preferences for $\texttt{NULL_FILTER}$ and $\texttt{HTML_SECTION_GROUP}$ as follows:

```
create index myindex on docs(htmlfile) indextype is ctxsys.context
  parameters('filter ctxsys.null_filter
  section group ctxsys.html section group');
```



For more information on section groups and indexing HTML documents, see "Section Group Types".



2.4.3 MAIL_FILTER

Use MAIL FILTER to transform RFC-822, RFC-2045 messages into indexable text.

The following limitations apply to the input:

- Documents must be US-ASCII
- Lines must not be longer than 1024 bytes
- Documents must be syntactically valid with regard to RFC-822.

Behavior for invalid input is not defined. Some deviations may be robustly handled by the filter without error. Others may result in a fetch-time or filter-time error.

- MAIL_FILTER Attributes
- MAIL FILTER Behavior
- About the Mail Filter Configuration File
- Mail_Filter Example

Note:

Starting with Oracle Database 18c, the use of MAIL_FILTER in Oracle Text is deprecated. Before adding email to the database, filter e-mails to indexable plain text, or to HTML.MAIL_FILTER is based on an obsolete email protocol, RFC-822. Modern email systems do not support RFC-822. There is no replacement.

2.4.3.1 MAIL FILTER Attributes

The MAIL FILTER has the attributes shown in Table 2-12.

Table 2-12 MAIL FILTER Attributes

Attribute	Attribute Value
INDEX_FIELDS	Specify a colon-separated list of fields to preserve in the output. These fields are transformed to tag markup. For example, if INDEX_FIELDS is set to "FROM":
	From: Scott Tiger
	becomes:
	<from>Scott Tiger</from>
	Only top-level fields are transformed in this way.
AUTO_FILTER_TIMEOUT	Specify a timeout value for the AUTO_FILTER filtering invoked by the mail filter. Default is 60. (Replaces the INSO_TIMEOUT attribute and is backward compatible with INSO_TIMEOUT.)
AUTO FILTER OUTPUT FORMATTING	Specify either TRUE or FALSE. Default is TRUE.
	This attribute replaces the previous INSO_OUTPUT_FORMATTING attribute. However, it has no effect in the current release.



Table 2-12 (Cont.) MAIL_FILTER Attributes

Attribute	Attribute Value
PART_FIELD_STYLE	Specify how fields occurring in lower-level parts and identified by the INDEX_FIELDS attribute should be transformed. The fields of
	the top-level message part identified by INDEX FIELDS are always
	transformed to tag markup (see the previous description of INDEX_FIELDS); PART_FIELD_STYLE controls the transformation of subsequent parts; for example, attached e-mails.
	Possible values include IGNORE (the default), in which the part fields are not included for indexing; TAG, in which the part field names are transformed to tags, as occurs with top-level part fields; FIELD, in which the part field names are preserved as fields, not as tags; and TEXT, in which the part field names are eliminated and only the field content is preserved for indexing. See "Mail_Filter Example" for an example of how PART_FIELD_STYLE works.

2.4.3.2 MAIL FILTER Behavior

This filter behaves in the following way for each document:

- Read and remove header fields
- Decode message body if needed, depending on Content-transfer-encoding field
- Take action depending on the Content-Type field value and the user-specified behavior specified in a mail filter configuration file. (See "About the Mail Filter Configuration File".)
 The possible actions are:
 - produce the body in the output text (INCLUDE). If no character set is encountered in the INCLUDE parts in the Content-Type header field, then Oracle defaults to the value specified in the character set column in the base table. Name your populated character set column in the parameter string of the CREATE INDEX command.
 - AUTO FILTER the body contents (AUTO FILTER directive).
 - remove the body contents from the output text (IGNORE)
- If no behavior is specified for the type in the configuration file, then the defaults are as follows:
 - text/*: produce body in the output text
 - application/*: AUTO FILTER the body contents
 - image/*, audio/*, video/*, model/*: ignore
- Multipart messages are parsed, and the mail filter applied recursively to each part. Each part is appended to the output.
- All text produced will be charset-converted to the database character set, if needed.

2.4.3.3 About the Mail Filter Configuration File

The MAIL_FILTER filter makes use of a mail filter configuration file, which contains directives specifying how a mail document should be filtered.



The mail filter configuration file is a editable text file. Here you can override default behavior for each Content-Type. The configuration file also contains IANA-to-Oracle Globalization Support character set name mappings.

The location of the file must be in <code>ORACLE_HOME/ctx/config</code>. The name of the file to use is stored in the new system parameter <code>MAIL_FILTER_CONFIG_FILE</code>. On install, this is set to drmailfl.txt, which has useful default contents.

Oracle recommends that you create your own mail filter configuration files to avoid overwrite by the installation of a new version or patch set. The mail filter configuration file should be in the database character set.

Mail File Configuration File Structure

The file has two sections, BEHAVIOR and CHARSETS. Indicate the start of the behavior section as follows:

```
[behavior]
```

Each line following starts with a mime type, then whitespace, then behavior specification. The MIME type can be a full TYPE/SUBTYPE or just TYPE, which will apply to all subtypes of that type. TYPE/SUBTYPE specification overrides TYPE specification, which overrides default behavior. Behavior can be INCLUDE, AUTO_FILTER, or IGNORE (see "MAIL_FILTER Behavior" for definitions). For instance:

```
application/zip IGNORE
application/msword AUTO_FILTER
model IGNORE
```

You cannot specify behavior for "multipart" or "message" types. If you do, such lines are ignored. Duplicate specification for a type replaces earlier specifications.

Comments can be included in the mail configuration file by starting lines with the # symbol.

The charset mapping section begins with

```
[charsets]
```

Lines consist of an IANA name, then whitespace, then an Oracle Globalization Support charset name, like:

```
US-ASCII US7ASCI
ISO-8859-1 WE8ISO8859P1
```

This file is the only way the mail filter gets the mappings. There are no defaults.

When you change the configuration file, the changes affect only the documents indexed after that point. You must flush the shared pool after changing the file.

2.4.3.4 Mail Filter Example

Suppose there is an e-mail with the following form, in which other e-mails with different subject lines are attached to this e-mail:

```
To: somebody@someplace
Subject: mainheader
Content-Type: multipart/mixed
. . .
Content-Type: text/plain
X-Ref: some_value
Subject: subheader 1
```



```
Content-Type: text/plain X-Control: blah blah blah Subject: subheader 2
```

Set INDEX FIELDS to be "Subject" and, initially, PART FIELD STYLE to IGNORE.

```
CTX_DDL.CREATE_PREFERENCE('my_mail_filt', 'mail_filter');
CTX_DDL_SET_ATTRIBUTE(my_mail_filt', 'INDEX_FILES', 'subject');
CTX_DDL.SET_ATTRIBUTE('my_mail_filt', 'PART_FIELD_STYLE', 'ignore');
```

Now when the index is created, the file will be indexed as follows:

```
<SUBJECT>mainheader</SUBJECT>
```

If PART_FIELD_STYLE is instead set to TAG, this becomes:

```
<SUBJECT>mainheader</SUBJECT>
<SUBJECT>subheader1</SUBJECT>
<SUBJECT>subheader2</SUBJECT>
```

If PART FIELD STYLE is set to FIELD instead, this is the result:

```
<SUBJECT>mainheader<SUBJECT>
SUBJECT:subheader1
SUBJECT:subheader2
```

Finally, if PART FIELD STYLE is instead set to TEXT, then the result is:

```
<SUBJECT>mainheader</SUBJECT>
subheader1
subheader2
```

2.4.4 USER FILTER

Use the USER FILTER type to specify an external filter for filtering documents in a column.

This section contains the following topics.

- USER FILTER Attributes
- Using USER FILTER with Charset and Format Columns
- USER_FILTER and Explicitly Bypassing Plain Text or HTML in Mixed Format Columns
- Character Set Conversion with USER_FILTER
- User Filter Example

2.4.4.1 USER_FILTER Attributes

USER FILTER has the following attribute:

Table 2-13 USER_FILTER Attribute

Attribute	Attribute Value
command	Specify the name of the filter executable.



WARNING:

The USER FILTER type introduces the potential for security threats. A database user granted the CTXAPP role could potentially use USER FILTER to load a malicious application. Therefore, the DBA must safeguard against any combination of input and output file parameters that would enable the named filter executable to compromise system security.

command

Specify the executable for the single external filter that is used to filter all text stored in a column. If more than one document format is stored in the column, then the external filter specified for command must recognize and handle all such formats.

The executable that you specify must exist in the <code>\$ORACLE HOME/ctx/bin directory</code> on UNIX, and in the %ORACLE HOME%/ctx/bin directory on Windows.

You must create your user-filter command with two parameters:

- The first parameter is the name of the input file to be read.
- The second parameter is the name of the output file to be written to.

If all the document formats are supported by AUTO FILTER, then use AUTO FILTER instead of USER FILTER, unless additional tasks besides filtering are required for the documents.

2.4.4.2 Using USER FILTER with Charset and Format Columns

USER FILTER bypasses documents that do not need to be filtered. Its behavior is sensitive to the values of the format and charset columns. In addition, USER FILTER performs character set conversion according to the charset column values.

2.4.4.3 USER FILTER and Explicitly Bypassing Plain Text or HTML in Mixed Format Columns

A mixed-format column is a text column containing more than one document format, such as a column that contains Microsoft Word, PDF, plain text, and HTML documents.

The USER FILTER executable can index mixed-format columns, automatically bypassing textual documents. However, if you prefer not to depend on the built-in bypass mechanism, you can explicitly tag your rows as text and cause the USER FILTER executable to ignore the row and not process the document in any way.

The format column in the base table enables you to specify the type of document contained in the text column. You can specify the following document types: TEXT, BINARY, and IGNORE. During indexing, the USER FILTER executable ignores any document typed TEXT, assuming the charset column is not specified. (The difference between a document with a TEXT format column type and one with an IGNORE type is that the TEXT document is indexed, but ignored by the filter, while the IGNORE document is not indexed at all. Use IGNORE to overlook documents such as image files, or documents in a language that you do not want to index. IGNORE can be used with any filter type.

To set up the USER FILTER bypass mechanism, you must create a format column in your base table. For example:

```
create table hdocs (
  id number primary key,
  fmt varchar2(10),
```



```
text varchar2(80)
);
```

Assuming you are indexing mostly Word documents, you specify BINARY in the format column to filter the Word documents. Alternatively, to have the USER_FILTER executable ignore an HTML document, specify TEXT in the format column.

For example, the following statements add two documents to the text table, assigning one format as BINARY and the other TEXT:

```
insert into hdocs values(1, 'binary', '/docs/myword.doc');
insert into hdocs values(2, 'text', '/docs/index.html');
commit;
```

Assuming that this file is named upcase.pl, create the filter preference as follows:

```
ctx_ddl.create_preference
   (
    preference_name => 'USER_FILTER_PREF',
    object_name => 'USER_FILTER'
   );

ctx ddl.set attribute ('USER FILTER PREF', 'COMMAND', 'upcase.pl');
```

To create the index, use CREATE INDEX and specify the format column name in the parameter string:

```
create index hdocsx on hdocs(text) indextype is ctxsys.context
  parameters ('datastore ctxsys.directory_datastore
  filter 'USER_FILTER_PREF'
  format column fmt');
```

If you do not specify TEXT or BINARY for the format column, BINARY is used.

2.4.4.4 Character Set Conversion with USER_FILTER

The <code>USER_FILTER</code> executable converts documents to the database character set when the document format column is set to <code>TEXT</code>. In this case, the <code>USER_FILTER</code> executable looks at the charset column to determine the document character set.

If the charset column value is not an Oracle Text character set name, the document is passed through without any character set conversion.

2.4.4.5 User Filter Example

The following example shows a Perl script to be used as the user filter. This script converts the input text file specified in the first argument to uppercase and writes the output to the location specified in the second argument.

```
#!/usr/local/bin/perl
open(IN, $ARGV[0]);
open(OUT, ">".$ARGV[1]);
while (<IN>)
{
   tr/a-z/A-Z/;
   print OUT;
}
```



```
close (IN);
close (OUT);
```

Assuming that this file is named upcase.pl, create the filter preference as follows:

```
begin
  ctx_ddl.create_preference
   (
      preference_name => 'USER_FILTER_PREF',
      object_name => 'USER_FILTER'
   );
  ctx_ddl.set_attribute
    ('USER_FILTER_PREF','COMMAND','upcase.pl');
end;
```

Create the index in SQL*Plus as follows:

```
create index user_filter_idx on user_filter ( docs )
  indextype is ctxsys.context
  parameters ('FILTER USER FILTER PREF');
```

2.4.5 PROCEDURE_FILTER

Use the PROCEDURE_FILTER type to filter your documents with a stored procedure. The stored procedure is called each time a document needs to be filtered.

This section contains the following topics.

- PROCEDURE FILTER Attributes
- PROCEDURE_FILTER Parameter Order
- PROCEDURE_FILTER Execute Requirements
- PROCEDURE FILTER Error Handling
- PROCEDURE_FILTER Preference Example

2.4.5.1 PROCEDURE_FILTER Attributes

Table 2-14 lists the attributes for PROCEDURE FILTER.

Table 2-14 PROCEDURE_FILTER Attributes

Attribute	Purpose	Allowable Values
procedure	Name of the filter stored procedure.	Any procedure. The procedure can be a PL/SQL stored procedure.
input_type	Type of input argument for stored procedure.	VARCHAR2, BLOB, CLOB, FILE
output_type	Type of output argument for stored procedure.	VARCHAR2, CLOB, FILE
rowid_parameter	Include rowid parameter?	TRUE/FALSE
format_parameter	Include format parameter?	TRUE/FALSE
charset_parameter	Include charset parameter?	TRUE/FALSE



procedure

Specify the name of the stored procedure to use for filtering. The procedure can be a PL/SQL stored procedure. The procedure can be a safe callout, or call a safe callout.

With the rowid_parameter, format_parameter, and charset_parameter set to FALSE, the procedure can have one of the following signatures:

```
PROCEDURE(IN BLOB, IN OUT NOCOPY CLOB)
PROCEDURE(IN CLOB, IN OUT NOCOPY CLOB)
PROCEDURE(IN VARCHAR, IN OUT NOCOPY CLOB)
PROCEDURE(IN BLOB, IN OUT NOCOPY VARCHAR2)
PROCEDURE(IN CLOB, IN OUT NOCOPY VARCHAR2)
PROCEDURE(IN VARCHAR2, IN OUT NOCOPY VARCHAR2)
PROCEDURE(IN BLOB, IN VARCHAR2)
PROCEDURE(IN CLOB, IN VARCHAR2)
PROCEDURE(IN VARCHAR2, IN VARCHAR2)
```

The first argument is the content of the unfiltered row, output by the datastore. The second argument is for the procedure to pass back the filtered document text. The procedure attribute is mandatory and has no default.

input_type

Specify the type of the input argument of the filter procedure. You can specify one of the following types:

Туре	Description
procedure	Name of the filter stored procedure.
input_type	Type of input argument for stored procedure.
output_type	Type of output argument for stored procedure.
rowid_parameter	Include rowid parameter?

The input type attribute is not mandatory. If not specified, then BLOB is the default.

output type

Specify the type of output argument of the filter procedure. You can specify one of the following types:

Туре	Description
CLOB	The output argument is IN OUT NOCOPY CLOB. Your procedure must write the filtered content to the CLOB passed in.
VARCHAR2	The output argument is IN OUT NOCOPY VARCHAR2. Your procedure must write the filtered content to the VARCHAR2 variable passed in.
FILE	The output argument must be IN VARCHAR2. On entering the filter procedure, the output argument is the name of a temporary file. The filter procedure must write the filtered contents to this named file. Using a FILE output type is useful only when the procedure is a safe callout, which can write to the file.

The <code>output_type</code> attribute is not mandatory. If not specified, then <code>CLOB</code> is the default.

rowid parameter

When you specify TRUE, the rowid of the document to be filtered is passed as the first parameter, before the input and output parameters.

For example, with INPUT_TYPE BLOB, OUTPUT_TYPE CLOB, and ROWID_PARAMETER TRUE, the filter procedure must have the signature as follows:

```
procedure (in rowid, in blob, in out nocopy clob)
```

This attribute is useful for when your procedure requires data from other columns or tables. This attribute is not mandatory. The default is FALSE.

format_parameter

When you specify TRUE, the value of the format column of the document being filtered is passed to the filter procedure before input and output parameters, but after the rowid parameter, if enabled.

Specify the name of the format column at index time in the parameters string, using the keyword 'format column <columname>'. The parameter type must be IN VARCHAR2. The format column value can be read by means of the rowid parameter, but this attribute enables a single filter to work on multiple table structures, because the format attribute is abstracted and does not require the knowledge of the name of the table or format column. FORMAT PARAMETERIS not mandatory. The default is FALSE.

charset_parameter

When you specify TRUE, the value of the charset column of the document being filtered is passed to the filter procedure before input and output parameters, but after the rowid and format parameter, if enabled.

Specify the name of the charset column at index time in the parameters string, using the keyword 'charset column <columnname>'. The parameter type must be IN VARCHAR2. The CHARSET PARAMETER attribute is not mandatory. The default is FALSE.

2.4.5.2 PROCEDURE FILTER Parameter Order

ROWID_PARAMETER, FORMAT_PARAMETER, and CHARSET_PARAMETER are all independent. The order is rowid, the format, then charset. However, the filter procedure is passed only the minimum parameters required.

For example, assume that INPUT_TYPE is BLOB and OUTPUT_TYPE is CLOB. If your filter procedure requires all parameters, then the procedure signature must be:

```
(id IN ROWID, format IN VARCHAR2, charset IN VARCHAR2, input IN BLOB, output IN OUT NOCOPY CLOB)
```

If your procedure requires only the ROWID, then the procedure signature must be:

(id IN ROWID, input IN BLOB, output IN OUT NOCOPY CLOB)

2.4.5.3 PROCEDURE FILTER Execute Requirements

To create an index using a PROCEDURE_FILTER preference, the index owner must have execute permission on the procedure.

2.4.5.4 PROCEDURE_FILTER Error Handling

The filter procedure can raise any errors needed through the normal PL/SQL raise_application_error facility. These errors are propagated to the CTX_USER_INDEX_ERRORS view or reported to the user, depending on how the filter is invoked.



2.4.5.5 PROCEDURE FILTER Preference Example

Consider a filter procedure CTXSYS.NORMALIZE that you define with the following signature:

```
PROCEDURE NORMALIZE(id IN ROWID, charset IN VARCHAR2, input IN CLOB, output IN OUT NOCOPY VARCHAR2);
```

To use this procedure as your filter, set up your filter preference as follows:

```
begin
ctx_ddl.create_preference('myfilt', 'procedure_filter');
ctx_ddl.set_attribute('myfilt', 'procedure', 'normalize');
ctx_ddl.set_attribute('myfilt', 'input_type', 'clob');
ctx_ddl.set_attribute('myfilt', 'output_type', 'varchar2');
ctx_ddl.set_attribute('myfilt', 'rowid_parameter', 'TRUE');
ctx_ddl.set_attribute('myfilt', 'charset_parameter', 'TRUE');
end:
```

2.5 Lexer Types

Use the lexer preference to specify the language of the text to be indexed. To create a lexer preference, you must use one of these lexer types.

- AUTO LEXER
- BASIC_LEXER
- MULTI_LEXER
- CHINESE_VGRAM_LEXER
- CHINESE LEXER
- JAPANESE_VGRAM_LEXER
- JAPANESE LEXER
- KOREAN MORPH LEXER
- USER LEXER
- WORLD_LEXER

2.5.1 AUTO_LEXER

Identifies the language being indexed by examining the content, and applies suitable options (including stemming) for that language. Works best where each document contains a single-language, and has at least a couple of paragraphs of text to aid identification.

Use the AUTO_LEXER type to index columns that contain documents of different languages. It performs language identification, word segmentation, document analysis, and stemming. The AUTO_LEXER also enables customization of these components. Although parts-of-speech information that is generated by the AUTO_LEXER is not exposed for your use, AUTO_LEXER uses it for context-sensitive or tagged stemming.

- AUTO_LEXER Language Support
- AUTO_LEXER Attributes Inherited from BASIC_LEXER
- AUTO_LEXER Language-Independent Attributes
- AUTO LEXER Language-Dependent Attributes



AUTO_LEXER Dictionary Attribute

2.5.1.1 AUTO LEXER Language Support

At index time, AUTO_LEXER automatically detects the language of the document, and tokenizes and stems the document appropriately.

AUTO LEXER Dictionary

To specify an AUTO_LEXER dictionary, use the name of the dictionary you created instead of the file name for the dictionary.

At query time, the language of the query is inherited from the query template. If the query template is not used, or if no language is specified in the query template, then the language of the query is inherited from the session language.



The dictionary data is not processed until the index or policy creation time or until the ALTER INDEX time. Errors in dictionary data format are caught at the index or policy creation time or at the ALTER INDEX time, and are reported as the "DRG-13710: Syntax Error in Dictionary" error.

AUTO_LEXER Component Versions

Starting with Oracle Database 23ai, the AUTO_LEXER component supports version ANL6, which is shipped with the Oracle Database installation by default.

The earlier version (ANL1) of the AUTO_LEXER component is available as an optional download patch. If you want to use version ANL1 to retain the prior language behavior for backward compatibility, then you can download ANL1 from My Oracle Support. After downloading the component, you must set Event 30579 Level 1048576 at the SYSTEM level.

Languages Distribution Model

- By default, Oracle Text ships language data files for only some of the languages supported
 for the AUTO_LEXER component. You can download data files for all other languages on
 demand from My Oracle Support using optional download patches. This language patch
 mechanism helps you control the installed languages and thus reduce the size of the
 database distribution for on-premises deployments.
- These language data files are included with the Oracle Database installation by default:

Arabic	Korean
Bokmal (Norwegian)	Nynorsk (Norwegian)
Catalan	Persian
Croatian	Polish
Czech	Portuguese



Danish	Romanian
Dutch	Russian
English	Serbian
Finnish	Simplified Chinese (see Note)
French	Slovak
German	Slovenian
Greek	Spanish
Hebrew	Swedish
Hungarian	Thai
Italian 	Traditional Chinese (see Note)
Japanese	Turkish

Note:

Due to the limitation of 30 characters for the string, Traditional Chinese must be specified as $trad_chinese$. Simplified Chinese must be specified as $simp_chinese$.

 You can download these language data files from My Oracle Support using optional download patches:

Afrikaans	Indonesian
Basque	Latvian
Belarusian	Lithuanian
Bulgarian	Macedonian
Estonian	Malay
Galician	Ukrainian
Hindi	Urdu
Icelandic	-



2.5.1.2 AUTO_LEXER Attributes Inherited from BASIC_LEXER

The following attributes are used in the same way and have the same effect on the AUTO_LEXER as their corresponding attributes in BASIC LEXER:

- printjoins
- skipjoins
- base letter
- base letter type
- override base letter
- mixed case
- alternate spelling



"BASIC_LEXER" and Table 2-19

2.5.1.3 AUTO_LEXER Language-Independent Attributes

These are the language-independent attributes that are supported for the <code>AUTO_LEXER</code> component.

Table 2-15 AUTO_LEXER Language-Independent Attributes

Attribute	Attribute Value	Description
language	characters (space-delimited string)	Specifies the possible languages of the input documents.
	oung)	If no language is specified, then AUTO_LEXER performs auto detection.
		If one language is specified, then the language is set manually and <code>AUTO_LEXER</code> does not perform auto detection.
		If more than one language is specified, then AUTO_LEXER performs auto detection but limits the detected language to be among the language set.
		Note: The automatic detection of language is statistically based and, thus, inherently imperfect.
deriv_stems	YES (default)	Specifies whether the derivational stemming should be used or not. Currently, derivational stemming is only available for
	NO (disabled)	English. Hence, the DERIV_STEMS has no effect in other languages.
		Also, when derivational stemming is performed, tagging and tag stemming is not used. As a result, the tagging and tagged stemming client dictionary has no effect on the stemming result.
german_decompound	YES (default, enabled for German only)	Specifies whether German de-compounding should be performed in the stemmer or not.
	NO (disabled)	performed in the stemmer of riot.



Table 2-15 (Cont.) AUTO_LEXER Language-Independent Attributes

Attribute	Attribute Value	Description
index_stems	YES (default) NO (disabled)	Specifies whether an index stemmer should be used. When set to YES, compound word stemming is automatically performed and compounds are always separated into their component stems. The stemmer that corresponds to the document language is used and the stemmer is always configured to maximize document recall. Note that this means that the stemmer attribute of BASIC_WORDLIST is ignored, and the stemmer used by the AUTO_LEXER is used during query to determine the stem of the given query term. When set to NO, queries with stem operators use the word list stemming to stem the tokens. If word list stemming is not available, then the stem operator is ignored.
base_letter	YES (enabled) NO (disabled)	Specify whether characters that have diacritical marks (umlauts, cedillas, acute accents, and so on) are converted to their base form before being stored in the Text index.
base_letter_type	SPECIFIC GENERIC (default)	The GENERIC value is the default and means that base letter transformation uses one transformation table that applies to all languages.
override_base_letter	TRUE FALSE (default)	When base_letter is enabled at the same time as alternate_spelling, it is sometimes necessary to override base_letter to prevent unexpected results from serial transformations.
mixed_case	YES (enabled) NO (disabled)	Specify whether the lexer leaves the tokens exactly as they appear in the text or converts the tokens to all uppercase. The default is NO (tokens are converted to all uppercase).
alternate_spelling	GERMAN (German alternate spelling) SWEDISH (Swedish alternate spelling) NONE (No alternate spelling, default)	Specifies whether alternate spelling should be used or not. The default is NONE. No alternate spelling is specified.
printjoins	characters	Specify the non alphanumeric characters that, when they appear anywhere in a word (beginning, middle, or end), are processed as alphanumeric and included with the token in the Text index. This includes printjoins that occur consecutively. See Basic Lexer "printjoins".
skipjoins	characters	Specify the non-alphanumeric characters that, when they appear within a word, identify the word as a single token; however, the characters are not stored with the token in the Text index. See Basic Lexer "skipjoins".



Table 2-15 (Cont.) AUTO_LEXER Language-Independent Attributes

Attribute	Attribute Value	Description
composite	YES (default)	Specify whether compound word stemming is enabled or
	is th	disabled for the supported languages text. The default value is YES (compound word stemming enabled). You can use this feature for all languages that are supported for AUTO_LEXER.
		When set to NO, words that are usually one entry in a dictionary are not split into composite stems, while words that are not dictionary entries are split into composite stems.
	To retrieve the indexed composite stems, you must enter a stem query. For example, \$bahnhof in German. The language of the wordlist stemmer must match the language of the composite stems.	
timeout	number	Specify the timeout value in seconds for auto_lexer tokenization.
		Use a number between 0 and 600. The default value is 300.

Related Topics

AUTO_LEXER Language Support

At index time, $AUTO_LEXER$ automatically detects the language of the document, and tokenizes and stems the document appropriately.

2.5.1.4 AUTO_LEXER Language-Dependent Attributes

These are the language-dependent attributes available in the AUTO_LEXER. The language variable in the attribute name refers to any of the supported language names.



Attribute names must not exceed 30 characters. Therefore, where the <language> variable is specified, the language name may need to be abbreviated in certain instances. For example, traditional_chinese should be abbreviated to trad chinese and simplified chinese should be abbreviated to simp chinese.

Table 2-16 AUTO_LEXER Language-Dependent Attributes

Attribute	Attribute Value	Description
<pre><language>_prefix_mor phemes</language></pre>	characters (space-delimited string)	Specifies the list of inflectional prefixes that, when enclosed by parentheses, are kept together with the base word. For example, <i>(re) analyze</i> .
<pre><language>_suffix_mor phemes</language></pre>	characters (space-delimited string)	Specifies the list of inflectional suffixes that, when enclosed by parentheses are kept together with the base word. For example, <i>file(s)</i> .
<pre><language>_punctuatio ns</language></pre>	characters (space-delimited string)	Specifies punctuation that breaks sentences.



Table 2-16 (Cont.) AUTO_LEXER Language-Dependent Attributes

Attribute	Attribute Value	Description
<pre><language>_non_sent_e nd_abbr</language></pre>	characters (space-delimited string)	Specifies abbreviations that do not end sentences.

Table 2-17 Default Values for AUTO_LEXER Language-Dependent Attributes

Attribute	Language	Default Value
<pre><language>_prefix_morphemes</language></pre>	All languages	None
<pre><language>_suffix_morphemes</language></pre>	English	s es er
<pre><language>_suffix_morphemes</language></pre>	Spanish	ba n s es
<pre><language>_suffix_morphemes</language></pre>	Portuguese	s es
<pre><language>_suffix_morphemes</language></pre>	German	in innen
<pre><language>_suffix_morphemes</language></pre>	French	ne e
<pre><language>_suffix_morphemes</language></pre>	All other languages	None
<pre><language>_punctuations</language></pre>	English	.?!
<pre><language>_punctuations</language></pre>	Catalan, Czech, Dutch, Greek, Hungarian, Polish, Romanian, Russian, Turkish	.?!
<pre><language>_punctuations</language></pre>	French, German, Italian, Korean, Portuguese, Spanish, Swedish	,?!
<pre><language>_punctuations</language></pre>	Japanese	
		.?!。?!:
<pre><language>_punctuations</language></pre>	Simplified Chinese Abbreviate to: simp_chinese	! ? ? !? . ? .
<pre><language>_punctuations</language></pre>	Traditional Chinese Abbreviate to: trad_chinese	! ? ? !? . ? ! (.
<pre><language>_non_sent_end_abbr</language></pre>	Polish, Romanian, Russian, Turkish	e.g. i.e. viz. a.k.a.
<pre><language>_non_sent_end_abbr</language></pre>	Catalan	R.D. pp.
<pre><language>_non_sent_end_abbr</language></pre>	Czech, Greek, Hungarian	e.g. i.e. viz. a.k.a.
<pre><language>_non_sent_end_abbr</language></pre>	Dutch	f.eks. f. eks. inkl. sr. skuesp. sekr. prof. mus. Irs. logr. kgl. insp. hr. hrs. gdr. frk. fr. forst. forf. fm. fmd. esq. d.æ d.æ. d.y. dr. dir. dept.chef civiling. bibl. ass. admn. adj. Skt. H.K.H.

Table 2-17 (Cont.) Default Values for AUTO_LEXER Language-Dependent Attributes

Attribute	Language	Default Value
<pre>_non_sent_end_abbr</pre>	English, Japanese, Simplified Chinese (abbreviate to simp_chinese), Traditional Chinese (abbreviate to trad_chinese)	e.g. i.e. viz. a.k.a. Adm. Br. Capt. Cdr. Cmdr. Col. Comdr. Comdt. Dr. Drs. Fr. Gen. Gov. Hon. Ins. Lieut. Lt. Maj. Messrs. Mdm. Mlle. Mlles. Mme. Mmes. Mr. Mrs. Ms. Pres. Prof. Profs. Pvt. Rep. Rev. Revd. Secy. Sen. Sgt. Sra. Srta. St. Ste.
<pre><language>_non_sent_end_abbr</language></pre>	French	cà-d. cf. e.g. ex. i.e. Pr. Prof. M. Mr. Mrs. Mme Mmes Mlle Mlles Mgr. MM. Lieut. Gén. Dr. Col.
<pre><language>_non_sent_end_abbr</language></pre>	German	ca. bzw. e.g. i.e. inkl. Fr. Frl. Mme. Mile. Mag. Stud. Tel. Hr. Hrn. apl.Prof. Prof.
<pre><language>_non_sent_end_abbr</language></pre>	Italian	e.g. i.e. pag. pagg. tel. T.V. N.H. N.D. comm. col. cav. cap. geom. gen. ing. jr. mr. mons. mar. magg. prof. prof.ssa prof.sse proff. pres. perito ind. p. p.i. sr. s.ten. sottoten. sig. serg. sen. segr. sac. ten. uff. vicepres. vesc. S.S. S.E. avv. app. amm. arch. on. dir. dott. dott.ssa dr. rag.
<pre><language>_non_sent_end_abbr</language></pre>	Korean	e.g. i.e. a.k.a. Dr. Mr. Mrs. Ms. Prof.
<pre><language>_non_sent_end_abbr</language></pre>	Portuguese	cf. Cf. e.g. E.g. i.é. I.é. p.ex. P.ex. pág. pag. Pág. Pag. tel. telef. Tel. Telef. sr. srs. sra. mr. eng. dr. dra. Dr. Dra. V.Ex. V.Exa. S. N. S. Mrs. Eng. Ex. Exa.
<pre><language>_non_sent_end_abbr</language></pre>	Spanish	e.g. i.e. ej. p.ej. pág. págs. tel. tfno. Fr. Ldo. Lda. Lic. Pbro. D. Dña. Dr. Dres. Dra. Dras. Dn. Mons. Rvdo. Sto. Sta. Sr. Srs. Srta. Srtas. Sres. Sra. Sras. Excmo. Excma. Ilmo. Ilma. Sto. Sta.
<pre><language>_non_sent_end_abbr</language></pre>	Swedish	inkl. prof. hrr. hr. Hrr. Hr. dr. Dr.

Examples for AUTO_LEXER Language-Dependent Attributes

Example 2-1 ctx_ddl.create_preference to associate a dictionary with an index

```
exec CTX_DDL.CREATE_PREFERENCE('A_LEX', 'AUTO_LEXER');
exec CTX_ANL. ADD_DICTIONARY('MY_ENGLISH', 'ENGLISH', lobloc);
select * from CTX_USR_ANL_DICTS;
exec CTX_DDL.SET_ATTRIBUTE('A_LEX', 'english_dictionary', 'MY_ENGLISH');
```

Example 2-2 < language > _prefix_morphemes

Example 2-3 < language > _suffix_morphemes



Example 2-4 < language > _ punctuations

Example 2-5 < language > _non_sentence_ending_abbrev

2.5.1.5 AUTO_LEXER Dictionary Attribute

The dictionary attribute is language-specific and is used to set the name of the language dictionary. The /dictionary attribute specifies one language dictionary for the supported languages as listed in Table 2-18.

The < language > dictionary attribute has the following behavior:

- The <language> value of the attribute specifies only the dictionary name, not the location. For example, dutch_dictionary specifies that the Dutch dictionary is to be used.
- The set_attribute method does not load the dictionary; it only records the dictionary name. Therefore, the dictionary must be at the specified location when the dictionary is needed. Otherwise, an error will be raised.

Table 2-18 Supported Languages for AUTO_LEXER Dictionary Attribute

Language Attribute	Language Attribute
Catalan	Korean
Czech	Polish
Dutch	Portuguese
English	Romanian
French	Russian
German	Simplified Chinese
Greek	Spanish
Hungarian	Swedish
Italian	Traditional Chinese
Japanese	Turkish

2.5.2 BASIC_LEXER

Extracts tokens from text in languages, such as English and most of the western European languages that use whitespace-delimited words.

Use the <code>BASIC_LEXER</code> type to identify tokens for creating Text indexes for English and all other supported whitespace-delimited languages. The <code>BASIC_LEXER</code> also enables base-letter conversion, composite word indexing, case-sensitive indexing and alternate spelling for whitespace-delimited languages that have extended character sets.

In English and French, you can use the BASIC LEXER to enable theme indexing.



Note:

Any processing that the lexer does to tokens before indexing (for example, removal of characters, and base-letter conversion) are also performed on query terms at query time. This ensures that the query terms match the form of the tokens in the Text index.

BASIC LEXER supports any database character set.

This section contains the following topics.

- BASIC LEXER Language Support
- BASIC LEXER Attributes
- · Stemming User-Dictionaries
- BASIC_LEXER Example

2.5.2.1 BASIC_LEXER Language Support

Oracle Text installs language data files for English by default. You can download data files for all other supported languages on demand from My Oracle Support.

Languages Distribution Model

Oracle Text utilizes installed data files for each supported language. Through cloud services, Oracle Text provides access to full versions of all supported languages. To reduce the installation footprint on disk for on-premises deployments, Oracle Text provides the following mechanism to control the number of downloaded languages:

- By default, full version of the English language data file is included with the Oracle
 Database installation. All other supported languages (apart from English) are distributed as
 optional download patches.
- Sample versions of some of the language data files are also included with the installation.
 You can utilize full versions of all these sample languages by downloading the required patches from My Oracle Support.

These languages are provided as both sample versions and download patches:

Catalan	Polish
Czech	Portuguese
Dutch	Romanian
French	Russian
German	Spanish
Greek	Swedish
Hungarian	Turkish



Italian -

 Some of the supported languages are distributed only as download patches with no sample included. You can utilize full versions of all these languages by downloading the required patches from My Oracle Support.

These languages are provided only as download patches:

Afrikaans	Icelandic
Arabic	Indonesian
Basque	Latvian
Belarusian	Lithuanian
Bokmal (Norwegian)	Macedonian
Bulgarian	Malay
Croatian	Nynorsk (Norwegian)
Danish	Persian (Farsi)
Estonian	Serbian
Finnish	Slovak
Galician	Slovenian
Hebrew	Ukrainian
Hindi	Urdu

2.5.2.2 BASIC_LEXER Attributes

These are the attributes supported for the BASIC LEXER component.

Table 2-19 BASIC_LEXER Attributes

Attribute	Attribute Value
continuation	characters
numgroup	characters
numjoin	characters
printjoins	characters
punctuations	characters
skipjoins	characters
startjoins	non alphanumeric characters that occur at the beginning of a token (string)



Table 2-19 (Cont.) BASIC_LEXER Attributes

Attribute	Attribute Value
endjoins	non alphanumeric characters that occur at the end of a token (string)
whitespace	characters (string)
newline	NEWLINE (\n)
	CARRIAGE_RETURN (\r)
base_letter	NO (disabled)
	YES (enabled)
base_letter_type	GENERIC (default)
	SPECIFIC
override_base_letter	TRUE
	FALSE (default)
mixed_case	NO (disabled)
	YES (enabled)



Table 2-19 (Cont.) BASIC_LEXER Attributes

Attribute	Attribute Value	
composite	YES (default; composite word indexing enabled)	
	Afrikaans	
	Arabic	
	Basque	
	Belarusian	
	Bokmal (Norwegian)	
	Bulgarian	
	Catalan	
	Croatian	
	Czech	
	Danish	
	Dutch	
	English	
	Estonian	
	Finnish	
	French	
	Galician	
	German	
	Greek	
	Hebrew	
	Hindi	
	Hungarian	
	Icelandic	
	Indonesian	
	Italian	
	Latvian	
	Lithuanian	
	Macedonian	
	Malay	
	Nynorsk (Norwegian)	
	Persian (Farsi)	
	Polish	
	Portuguese	
	Romanian	
	Russian	
	Serbian	
	Slovak	
	Slovenian	
	Spanish	
	Swedish	
	Turkish	
	Ukrainian	

Table 2-19 (Cont.) BASIC_LEXER Attributes

Attribute	Attribute Value
	Urdu



Table 2-19 (Cont.) BASIC_LEXER Attributes

Attribute	Attribute Value
index_stems	NONE
Use the numeric value in a string or the string value.	Afrikaans
	Arabic
	Basque
	Belarusian
	Bokmal (Norwegian)
	Bulgarian
	Catalan
	Croatian
	Czech
	Danish
	Derivational
	Dutch
	English
	Estonian
	Finnish
	French
	Galician
	German
	Greek
	Hebrew
	Hindi
	Hungarian
	Icelandic
	Indonesian
	Italian
	Latvian
	Lithuanian
	Macedonian
	Malay
	Nynorsk (Norwegian)
	Persian (Farsi)
	Polish
	Portuguese
	Romanian
	Russian
	Serbian
	Slovak
	Slovenian
	Spanish
	Swedish
	Turkish

Table 2-19 (Cont.) BASIC_LEXER Attributes

Attribute	Attribute Value	
	Ukrainian	
	Urdu	

Note:

- Apart from English, all other languages are either provided in sample sizes (with full data files available for download) or as optional download patches.
- De-compounding word stemming is automatically performed when index_stems is set to Swedish or Dutch values.
- In previous releases, index_stems attributes with the _New suffix used to enable a new stemmer for maintaining backward compatibility with the old stemmer. Starting with Oracle Database 23ai, the old stemmer has been removed, making the _New suffix redundant. For example, English_New is equivalent to English.

index_themes	YES (enabled) NO (disabled, default)
index_text	YES (enabled, default) NO (disabled)
prove_themes	YES (enabled, default) NO (disabled)
theme_language	AUTO (default) (any Globalization Support language)
alternate_spelling	German (German alternate spelling) Danish (Danish alternate spelling) Swedish (Swedish alternate spelling) NONE (No alternate spelling, default)
new_german_spelling	YES NO (default)

continuation

Specify the characters that indicate a word continues on the next line and should be indexed as a single token. The most common continuation characters are hyphen '-' and backslash '\'.

numgroup

Specify a single character that, when it appears in a string of digits, indicates that the digits are groupings within a larger single unit.

For example, comma ',' might be defined as a numgroup character because it often indicates a grouping of thousands when it appears in a string of digits.

numjoin

Specify the characters that, when they appear in a string of digits, cause Oracle Text to index the string of digits as a single unit or word.

For example, period '.' can be defined as a numjoin character because it often serves as a decimal point when it appears in a string of digits.



The default values for numjoin and numgroup are determined by the globalization support initialization parameters that are specified for the database. In general, a value need not be specified for either numjoin or numgroup when

creating a lexer preference for BASIC LEXER.

printjoins

Specify the non alphanumeric characters that, when they appear anywhere in a word (beginning, middle, or end), are processed as alphanumeric and included with the token in the Text index. This includes printjoins that occur consecutively.

For example, if the hyphen '-' and underscore '_' characters are defined as printjoins, terms such as pseudo-intellectual and _file_ are stored in the Text index as pseudo-intellectual and _file_.

Note:

If a printjoins character is also defined as a punctuations character, the character is only processed as an alphanumeric character if the character immediately following it is a standard alphanumeric character or has been defined as a printjoins or skipjoins character.

punctuations

Specify a list of non-alphanumeric characters that, when they appear at the end of a word, indicate the end of a sentence. The defaults are period '.', question mark '?', and exclamation point '!'.

Characters that are defined as punctuations are removed from a token before text indexing. However, if a punctuations character is also defined as a printjoins character, then the character is removed only when it is the last character in the token.

For example, if the period (.) is defined as both a printjoins and a punctuations character, then the following transformations take place during indexing and querying as well:

Token	Indexed Token
.doc	.doc
dog.doc	dog.doc
dogdoc	dogdoc
dog.	dog



Token	Indexed Token
dog	dog

In addition, BASIC_LEXER use punctuations characters in conjunction with newline and whitespace characters to determine sentence and paragraph delimiters for sentence/paragraph searching.

skipjoins

Specify the non-alphanumeric characters that, when they appear within a word, identify the word as a single token; however, the characters are not stored with the token in the Text index.

For example, if the hyphen character '-' is defined as a skipjoins, then the word pseudo-intellectual is stored in the Text index as pseudointellectual.



Printjoins and skipjoins are mutually exclusive. The same characters cannot be specified for both attributes.

startjoins/endjoins

For startjoins, specify the characters that when encountered as the first character in a token explicitly identify the start of the token. The character, as well as any other startjoins characters that immediately follow it, is included in the Text index entry for the token. In addition, the first startjoins character in a string of startjoins characters implicitly ends the previous token.

For endjoins, specify the characters that when encountered as the last character in a token explicitly identify the end of the token. The character, as well as any other startjoins characters that immediately follow it, is included in the Text index entry for the token. The following rules apply to both startjoins and endjoins:

- The characters specified for startjoins/endjoins cannot occur in any of the other attributes for BASIC_LEXER.
- startjoins/endjoins characters can occur only at the beginning or end of tokens

Printjoins differ from endjoins and startjoins in that position does not matter. For example, \$35 will be indexed as one token if \$ is a startjoin or a printjoin, but as two tokens if it is defined as an endjoin.

whitespace

Specify the characters that are treated as blank spaces between tokens. BASIC_LEXER uses whitespace characters in conjunction with punctuations and newline characters to identify character strings that serve as sentence delimiters for sentence and paragraph searching. The predefined default values for whitespace are space and tab. These values cannot be changed. Specifying characters as whitespace characters adds to these defaults.

newline

Specify the characters that indicate the end of a line of text. BASIC_LEXER uses newline characters in conjunction with punctuations and whitespace characters to identify character strings that serve as paragraph delimiters for sentence and paragraph searching. The only valid values for newline are NEWLINE and CARRIAGE_RETURN (for carriage returns). The default is NEWLINE.



base letter

Specify whether characters that have diacritical marks (umlauts, cedillas, acute accents, and so on) are converted to their base form before being stored in the Text index. The default is NO (base-letter conversion disabled). For more information on base-letter conversions and base letter type, see Base-Letter Conversion.

base_letter_type

Specify GENERIC or SPECIFIC.

The GENERIC value is the default and means that base letter transformation uses one transformation table that applies to all languages. For more information on base-letter conversions and base letter type, see "Base-Letter Conversion".

override base letter

When base_letter is enabled at the same time as alternate_spelling, it is sometimes necessary to override base_letter to prevent unexpected results from serial transformations. See "Overriding Alternative Spelling Features". Default is FALSE.

mixed case

Specify whether the lexer leaves the tokens exactly as they appear in the text or converts the tokens to all uppercase. The default is NO (tokens are converted to all uppercase).



Oracle Text ensures that word queries match the case sensitivity of the index being queried. As a result, if you enable case sensitivity for your Text index, queries against the index are always case sensitive.

composite

Specify whether composite word indexing is disabled or enabled for the supported languages text. The default value is YES (composite word indexing enabled). You can use this feature for all languages that are supported for BASIC LEXER.

Words that are usually one entry in a dictionary are not split into composite stems, while words that are not dictionary entries are split into composite stems.

To retrieve the indexed composite stems, you must enter a stem query. For example, \$bahnhof in German. The language of the wordlist stemmer must match the language of the composite stems.

Related Topics

BASIC_LEXER Language Support
 Oracle Text installs language data files for English by default. You can download data files for all other supported languages on demand from My Oracle Support.

2.5.2.3 Stemming User-Dictionaries

You can create a user-dictionary for your own language to customize how words are decomposed.

Table 2-20 Stemming User-Dictionaries

Dictionary	Stemmer
\$ORACLE_HOME/ctx/data/frlx/drfr.dct	French
<pre>\$ORACLE_HOME/ctx/data/delx/drde.dct</pre>	German



Table 2-20 (Cont.) Stemming User-Dictionaries

Dictionary	Stemmer
\$ORACLE_HOME/ctx/data/nllx/drnl.dct	Dutch
<pre>\$ORACLE_HOME/ctx/data/itlx/drit.dct</pre>	Italian
<pre>\$ORACLE_HOME/ctx/data/eslx/dres.dct</pre>	Spanish
<pre>\$ORACLE_HOME/ctx/data/enlx/dren.dct</pre>	English and Derivational

Stemming user-dictionaries are not supported for languages other than those listed in Table 2-20.

The format for the user dictionary is as follows:

output term <tab> input term

The individual parts of the decomposed word must be separated by the # character. The following example entries are for the German word *Hauptbahnhof*:

Hauptbahnhof

Haupt#Bahnhof

Hauptbahnhofes<tab>Haupt#Bahnhof

Hauptbahnhof

Haupt#Bahnhof

Haupt#Bahnhof

index_themes

Specify YES to index theme information in English or French. This makes ABOUT queries more precise. The <code>index_themes</code> and <code>index_text</code> attributes cannot both be NO. The default is NO. You can set this parameter to <code>TRUE</code> for any index type. To enter an <code>ABOUT</code> query with <code>CATSEARCH</code>, use the query template with <code>CONTEXT</code> grammar.

prove themes

Specify YES to prove themes. Theme proving attempts to find related themes in a document. When no related themes are found, parent themes are eliminated from the document. While theme proving is acceptable for large documents, short text descriptions with a few words rarely prove parent themes, resulting in poor recall performance with ABOUT queries. Theme proving results in higher precision and less recall (less rows returned) for ABOUT queries. For higher recall in ABOUT queries and possibly less precision, you can disable theme proving. Default is YES.

The prove themes attribute is supported for CONTEXT and CTXRULE indexes.

theme_language

Specify which knowledge base to use for theme generation when <code>index_themes</code> is set to YES. When <code>index_themes</code> is NO, setting this parameter has no effect on anything.

Specify any globalization support language or AUTO. You must have a knowledge base for the language you specify. This release provides a knowledge base in only English and French. In other languages, you can create your own knowledge base.



"Adding a Language-Specific Knowledge Base" in Oracle Text Utilities .



The default is AUTO, which instructs the system to set this parameter according to the language of the environment.

index stems

Specify the stemmer to use for stem indexing. Choose one of the following stemmers: NONE, Arabic, Bokmal (Norwegian), Catalan, Croatian, Czech, Danish, Derivational, Dutch, English, Finnish, French, German, Hebrew, Hungarian, Italian, Nynorsk (Norwegian), Polish, Portuguese, Romanian, Slovak, Slovenian, Spanish, and Swedish

Tokens are stemmed to a single base form at index time in addition to the normal forms. Indexing stems enables better query performance for stem (\$) queries, such as \$computed.



If the <code>index_stems</code> attribute is set to one of the languages with ID 8 to 33, which are listed Table 2-19, then the <code>stemmer</code> attribute of <code>BASIC_WORDLIST</code> will be ignored and the stemmer used by the <code>BASIC_LEXER</code> will be used during query to determine the stem of the given query term.

index_text

Specify YES to index word information. The $index_themes$ and $index_text$ attributes cannot both be NO.

The default is YES.

alternate_spelling

Specify either German, Danish, or Swedish to enable the alternate spelling in one of these languages. Enabling alternate spelling enables you to query a word in any of its alternate forms

Alternate spelling is off by default; however, in the language-specific scripts that Oracle provides in admin/defaults (drdefd.sql for German, drdefdk.sql for Danish, and drdefs.sql for Swedish), alternate spelling is turned on. If your installation uses these scripts, then alternate spelling is on. However, you can specify NONE for no alternate spelling. For more information about the alternate spelling conventions Oracle Text uses, see Alternate Spelling.

new german spelling

Specify whether the queries using the BASIC_LEXER return both traditional and reformed (new) spellings of German words. If new_german_spelling is set to YES, then both traditional and new forms of words are indexed. If it is set to NO, then the word will be indexed only as it as provided in the query. The default is NO.



"New German Spelling"

2.5.2.4 BASIC_LEXER Example

The following example sets printjoin characters and disables theme indexing with the BASIC LEXER:

```
begin
ctx_ddl.create_preference('mylex', 'BASIC_LEXER');
```

```
ctx_ddl.set_attribute('mylex', 'printjoins', '_-');
ctx_ddl.set_attribute ( 'mylex', 'index_themes', 'NO');
ctx_ddl.set_attribute ( 'mylex', 'index_text', 'YES');
end:
```

To create the index with no theme indexing and with printjoin characters set as described, enter the following statement:

```
create index myindex on mytable ( docs )
indextype is ctxsys.context
parameters ( 'LEXER mylex' );
```

2.5.3 MULTI LEXER

Requires a Language column in the table that identifies the language for each document. Each language has an associated sub-lexer, defined by the user. This lexer has no attributes.

Use MULTI_LEXER to index text columns that contain documents of different languages. For example, use this lexer to index a text column that stores English, German, and Japanese documents.

You must have a LANGUAGE column in your base table. To index multi-language tables, specify the LANGUAGE column when you create the index. You must also specify the language at query time (through Session settings or a Language settings in a query template), and the queries only look for documents that are indexed using the current language.

Create a multi-lexer preference with CTX_DDL.CREATE_PREFERENCE. Add language-specific lexers to the multi-lexer preference with the CTX_DDL.ADD_SUB_LEXER procedure.

During indexing, the MULTI_LEXER examines each row's language column value and switches in the language-specific lexer to process the document.



If you drop the language column from a multi-lexer indexed table, you must also drop the index and rebuild it.

The WORLD_LEXER lexer also performs multi-language indexing, but without the need for separate LANGUAGE columns (that is, it has automatic language detection). For more on WORLD LEXER, see "WORLD_LEXER".

This section contains the following topics.

- MULTI_LEXER Restriction
- MULTI LEXER Multi-language Stoplists
- MULTI_LEXER Example
- MULTI_LEXER and Querying Multi-Language Tables

2.5.3.1 MULTI LEXER Restriction

MULTI_LEXER must have a sublexer specified for different languages. If you already know the language, you can use BASIC_LEXER as the sublexer. If the language is not known, then you use AUTO_LEXER instead of MULTI_LEXER. Hence, using AUTO_LEXER as a sublexer of MULTI_LEXER is not useful and it is disabled.



Thus, the following statements will not work and throw error DRG-13003.

```
exec ctx_ddl.create_preference ('multilexer', 'MULTI_LEXER');
exec ctx_ddl.create_preference('autolexer', AUTO_LEXER);
exec ctx_ddl.add sub_lexer('multilexer', 'GERMAN', 'autolexer');
```

2.5.3.2 MULTI_LEXER Multi-language Stoplists

When you use the MULTI LEXER, you can also use a multi-language stoplist for indexing.



"Multi-Language Stoplists".

2.5.3.3 MULTI LEXER Example

Create the multi-language table with a primary key, a text column, and a language column as follows:

```
create table globaldoc (
   doc_id number primary key,
   lang varchar2(3),
   text clob
);
```

Assume that the table holds mostly English documents, with the occasional German or Japanese document. To handle the three languages, you must create three sub-lexers, one for English, one for German, and one for Japanese:

```
ctx_ddl.create_preference('english_lexer', 'basic_lexer');
ctx_ddl.set_attribute('english_lexer', 'index_themes', 'yes');
ctx_ddl.set_attribute('english_lexer', 'theme_language', 'english');
ctx_ddl.create_preference('german_lexer', 'basic_lexer');
ctx_ddl.set_attribute('german_lexer', 'composite', 'german');
ctx_ddl.set_attribute('german_lexer', 'mixed_case', 'yes');
ctx_ddl.set_attribute('german_lexer', 'alternate_spelling', 'german');
ctx_ddl.create_preference('japanese_lexer', 'japanese_vgram_lexer');
```

Create the multi-lexer preference:

```
ctx ddl.create preference('global lexer', 'multi lexer');
```

Because the stored documents are mostly English, make the English lexer the default using CTX_DDL.ADD_SUB_LEXER:

```
ctx_ddl.add_sub_lexer('global_lexer','default','english_lexer');
```

Now add the German and Japanese lexers in their respective languages with CTX_DDL.ADD_SUB_LEXER procedure. Also assume that the language column is expressed in the standard ISO 639-2 language codes, so add those as alternative values.

```
ctx_ddl.add_sub_lexer('global_lexer','german','german_lexer','ger');
ctx_ddl.add_sub_lexer('global_lexer','japanese','japanese_lexer','jpn');
```

Now create the index globalx, specifying the multi-lexer preference and the language column in the parameter clause as follows:

create index globalx on globaldoc(text) indextype is ctxsys.context
parameters ('lexer global lexer language column lang');

2.5.3.4 MULTI_LEXER and Querying Multi-Language Tables

At query time, the multi-lexer examines the language setting and uses the sub-lexer preference for that language to parse the query.

If the language is not set, then the default lexer is used. Otherwise, the query is parsed and run as usual. The index contains tokens from multiple languages, so such a query can return documents in several languages. To limit your query to a given language, use a structured clause on the language column.

If the language column is set to AUTO, then the multi-lexer detects the language of the document for the supported languages shown in Table 2-21.

Table 2-21 Languages Supported for MULTI_LEXER Auto-detection

Language	Language
Arabic	Japanese
Bokmal (Norwegian)	Korean
Catalan	Latin Serbian
Croatian	Nynorsk (Norwegian)
Czech	Polish
Danish	Portuguese
Dutch	Romanian
English	Russian
German	Slovak
Greek	Swedish
Hebrew	Thai
Hungarian	Traditional Chinese
Italian	Turkish

2.5.4 CHINESE_VGRAM_LEXER

Extracts tokens in Chinese text for creating Oracle Text indexes.

Table 2-22 CHINESE VGRAM LEXER Attributes

Attribute	Attribute Value	
mixed_case_ASCII7	Enable mixed-case (upper- and lower-case) searches of ASCII7 text (for example, $\it cat$ and $\it Cat$). Allowable values are YES and NO (default).	

You can use this lexer if your database uses one of the following character sets:

- AL32UTF8
- ZHS16CGB231280



- ZHS16GBK
- ZHS32GB18030
- ZHT32EUC
- ZHT16BIG5
- ZHT32TRIS
- ZHT16HKSCS
- ZHT16MSWIN950
- UTF8

2.5.5 CHINESE_LEXER

Identifies tokens in traditional and simplified Chinese text for creating Oracle Text indexes.

The CHINESE LEXER type offers the following benefits over the CHINESE VGRAM LEXER:

- generates a smaller index
- better query response time
- generates real word tokens resulting in better query precision
- supports stop words

Because the <code>CHINESE_LEXER</code> uses a different algorithm to generate tokens, indexing time is longer than with <code>CHINESE_VGRAM_LEXER</code>.

You can use this lexer if your database character is one of the Chinese or Unicode character sets supported by Oracle.

The CHINESE LEXER has the following attribute:

Table 2-23 CHINESE_LEXER Attributes

Attribute	Attribute Value
mixed_case_ASCII7	Enable mixed-case (upper- and lower-case) searches of ASCII7 text (for example, <i>cat</i> and <i>Cat</i>). Allowable values are YES and NO (default).

You can modify the existing lexicon (dictionary) used by the Chinese lexer, or create your own Chinese lexicon, with the ctxlc command.



"Lexical Compiler (ctxlc)" in Oracle Text Utilities

2.5.6 JAPANESE_VGRAM_LEXER

Identifies tokens in Japanese for creating Oracle Text indexes. This lexer supports the stem (\$) operator.



Table 2-24 JAPANESE_VGRAM_LEXER Attributes

Attribute	Attribute Value	
delimiter	Specify whether to consider certain Japanese blank characters, such as a full-width forward slash or a full-width middle dot, as part of the indexed token. ALL considers these characters as part of the token while NONE ignores them. The default is NONE.	
mixed_case_ASCII7	Enable mixed-case (upper- and lower-case) searches of ASCII7 text (for example, cat and Cat). Allowable values are YES and NO (default).	
bigram	Specify TRUE to enable the <i>bigram</i> mode for the Japanese VGRAM lexer. In the <i>bigram</i> mode, the Japanese queries run faster because only 2-gram tokens are generated, thus avoiding the internal wildcard search. But, in the bigram mode, the index size needs to be increased to accommodate the large number of tokens. Enable the <i>bigram</i> mode, if the performance of queries is of higher importance to you than the disk space. Default is FALSE.	
printjoins	Specify the non alphanumeric characters that, when they appear anywhere in a word (beginning, middle, or end), are processed as alphanumeric and included with the token in the Text index. This includes printjoins that occur consecutively. See Basic Lexer "printjoins".	
skipjoins	Specify the non-alphanumeric characters that, when they appear within a word, identify the word as a single token; however, the characters are not stored with the token in the Text index. See Basic Lexer "skipjoins".	

You can use this lexer if your database uses one of the following character sets:

- JA16SJIS
- JA16EUC
- UTF8
- AL32UTF8
- JA16EUCTILDE
- JA16EUCYEN
- JA16SJISTILDE
- JA16SJISYEN

Rules for PRINTJOIN and SKIPJOIN Characters

- Only non-alphanumeric ASCII characters *that do not include* any Chinese, Japanese, or Korean characters *or* any full-width non-alphanumeric characters are accepted.
- You can specify a single non-alphanumeric character or multiple non-alphanumeric characters at a time.
- The printjoin/skipjoin will be ignored if you enter any characters that are not allowed. This includes alphanumeric characters, CJK Chinese, Japanese, Korean characters or full-width non-alphanumeric characters.
- In case of duplicate non-alphanumeric characters, duplicate entries will be ignored.



Examples

Example 2-6 Using Printjoins with JAPANESE_VGRAM_LEXER

This example defines the hyphen and underscore characters as printjoins thereby indicating that these characters must be included with the token in the Text index. Therefore, words such as web-site or web_site as indexed as web-site and web_site. Queries that search for website will not return documents containing web-site or web_site.

```
ctx_ddl.create_preference('mylex', 'JAPANESE_VGRAM_LEXER');
ctx_ddl.set_attribute('mylex', 'printjoins', '_-');
```

Example 2-7 Using Skipjoins with JAPANESE_VGRAM_LEXER

This example defines the hyphen and underscore characters as skipjoins thereby indicating that these characters must *not* be included with the token in the Text index. Therefore, words such as *web-site* or *web_site* as indexed as *website*. Queries that search for *website* will return documents containing *web-site* or *web_site*.

```
ctx_ddl.create_preference('mylex', 'JAPANESE_VGRAM_LEXER');
ctx ddl.set attribute('mylex', 'skipjoins', ' -');
```

2.5.7 JAPANESE LEXER

Identifies tokens in Japanese for creating Oracle Text indexes. Offers advantages over <code>JAPANESE_VGRAM_LEXER</code>, such as generates a smaller index, has a better query response time, and generates real word tokens resulting in better query precision.

The JAPANESE_LEXER type supports the stem (\$) operator. Because the JAPANESE_LEXER uses a new algorithm to generate tokens, indexing time is longer than with JAPANESE VGRAM LEXER.

You can modify the existing lexicon (dictionary) used by the Japanese lexer, or create your own Japanese lexicon, with the ctxlc command.



"Lexical Compiler (ctxlc)" in Oracle Text Utilities

This lexer has the following attributes:

Table 2-25 JAPANESE_LEXER Attributes

Attribute	Attribute Value
delimiter	Specify NONE or ALL to ignore certain Japanese blank characters, such as a full-width forward slash or a full-width middle dot. Default is NONE.
mixed_case_ASCII7	Enable mixed-case (upper- and lower-case) searches of ASCII7 text (for example, <i>cat</i> and <i>Cat</i>). Allowable values are YES and NO (default).

The Japanese lexer supports the following character sets:

JA16SJIS



- JA16EUC
- UTF8
- AL32UTF8
- JA16EUCTILDE
- JA16EUCYEN
- JA16SJISTILDE
- JA16SJISYEN

When you specify JAPANESE_LEXER for creating text index, the JAPANESE_LEXER resolves a sentence into words.

For example, the following compound word (natural language institute)

'自然言語処理'

is indexed as three tokens:

'自然','言語','処理'

To resolve a sentence into words, the internal dictionary is referenced. When a word cannot be found in the internal dictionary, Oracle Text uses the JAPANESE VGRAM LEXER to resolve it.

2.5.8 KOREAN_MORPH_LEXER

Identifies tokens in Korean text for creating Oracle Text indexes.

This section contains the following topics.

- KOREAN_MORPH_ LEXER Dictionaries
- KOREAN_MORPH_ LEXER Unicode Support
- KOREAN_MORPH_LEXER Attributes
- KOREAN_MORPH_ LEXER Limitations
- KOREAN_MORPH_LEXER Example: Setting Composite Attribute

2.5.8.1 KOREAN MORPH LEXER Dictionaries

The KOREAN MORPH LEXER uses four dictionaries:



Table 2-26 KOREAN_MORPH_LEXER Dictionaries

Dictionary	File
System	\$ORACLE_HOME/ctx/data/kolx/drk2sdic.dat
Grammar	<pre>\$ORACLE_HOME/ctx/data/kolx/drk2gram.dat</pre>
Stopword	<pre>\$ORACLE_HOME/ctx/data/kolx/drk2xdic.dat</pre>
User-defined	<pre>\$ORACLE_HOME/ctx/data/kolx/drk2udic.dat</pre>

The grammar, user-defined, and stopword dictionaries should be written using the KSC 5601 or MSWIN949 character sets. You can modify these dictionaries using the defined rules. The system dictionary must not be modified.

You can add unregistered words to the user-defined dictionary file. The rules for specifying new words are in the file.

You can use Korean Morph Lexer if your database uses one of the following character sets:

- KO16KSC5601
- KO16MSWIN949
- UTF8
- AL32UTF8

The KOREAN MORPH LEXER enables mixed-case searches.

2.5.8.2 KOREAN_MORPH_ LEXER Unicode Support

The KOREAN_MORPH_LEXER has the following Unicode support:

- Words in non-KSC5601 Korean characters defined in Unicode
- Supplementary characters



For information on supplementary characters, see the *Oracle Database Globalization* Support Guide

Some Korean documents may have non-KSC5601 characters in them. As the KOREAN_MORPH_LEXER can recognize all possible 11,172 Korean (Hangul) characters, such documents can also be interpreted by using the UTF8 or AL32UTF8 character sets.

Use the AL32UTF8 character set for your database to extract surrogate characters. By default, the KOREAN_MORPH_LEXER extracts all series of surrogate characters in a document as one token for each series.

Limitations on Korean Unicode Support

For conversion from Hanja to Hangul (Korean), the KOREAN_MORPH_LEXER supports only the 4,888 Hanja characters defined in KSC5601.



2.5.8.3 KOREAN MORPH LEXER Attributes

When you use the KOREAN MORPH LEXER, you can specify the following attributes:

Table 2-27 KOREAN_MORPH_LEXER Attributes

Attribute	Attribute Value	
verb_adjective	Specify TRUE or FALSE to index verbs, adjectives, and adverbs. Default is FALSE.	
one_char_word	Specify TRUE or FALSE to index one syllable. Default is FALSE.	
number	Specify TRUE or FALSE to index number. Default is FALSE.	
user_dic	Specify TRUE or FALSE to index user dictionary. Default is TRUE.	
stop_dic	Specify TRUE of FALSE to use stop-word dictionary. Default is TRUE. The stop-word dictionary belongs to KOREAN_MORPH_LEXER.	
composite	Specify indexing style of composite noun.	
	Specify COMPOSITE_ONLY to index only composite nouns.	
	Specify NGRAM to index all noun components of a composite noun.	
	Specify COMPONENT_WORD to index single noun components of composite nouns as well as the composite noun itself. Default is COMPONENT_WORD.	
	"KOREAN_MORPH_LEXER Example: Setting Composite Attribute" describes the difference between NGRAM and COMPONENT_WORD.	
morpheme	Specify TRUE or FALSE for morphological analysis. If set to FALSE, tokens are created from the words that are divided by delimiters such as white space in the document. Default is TRUE.	
to_upper	Specify TRUE or FALSE to convert English to uppercase. Default is TRUE.	
hanja	Specify TRUE to index hanja characters. If set to FALSE, hanja characters are converted to hangul characters. Default is FALSE.	
long_word	Specify TRUE to index long words that have more than 16 syllables in Korean. Default is FALSE.	
japanese	Specify TRUE to index Japanese characters in Unicode (only in the 2-byte area). Default is FALSE.	
english	Specify TRUE to index alphanumeric strings. Default is TRUE.	

2.5.8.4 KOREAN_MORPH_ LEXER Limitations

Sentence and paragraph sections are not supported with the KOREAN_MORPH_LEXER.

2.5.8.5 KOREAN MORPH LEXER Example: Setting Composite Attribute

Use the composite attribute to control how composite nouns are indexed.

NGRAM Example

When you specify NGRAM for the composite attribute, composite nouns are indexed with all possible component tokens. For example, the following composite noun (*information processing institute*)

'정보처리학회'.



is indexed as six tokens:

'정보', '처리', '학회', '정보처리',

'처리학회', '정보처리학회'

Specify NGRAM indexing as follows:

```
begin
ctx_ddl.create_preference('my_lexer','KOREAN_MORPH_LEXER');
ctx_ddl.set_attribute('my_lexer','COMPOSITE','NGRAM');
end
```

To create the index:

```
create index koreanx on korean(text) indextype is ctxsys.context
parameters ('lexer my_lexer');
```

COMPONENT_WORD Example

When you specify COMPONENT_WORD for the composite attribute, composite nouns and their components are indexed. For example, the following composite noun (*information processing institute*)

```
'정보처리학회',
```

is indexed as four tokens:

```
'정보', '처리', '학회'
```

Specify COMPONENT_WORD indexing as follows:

```
begin
ctx_ddl.create_preference('my_lexer','KOREAN_MORPH_LEXER');
ctx_ddl.set_attribute('my_lexer','COMPOSITE','COMPONENT_WORD');
end
```

To create the index:

```
create index koreanx on korean(text) indextype is ctxsys.context
parameters ('lexer my_lexer');
```

2.5.9 USER_LEXER

Lexer you create to index a particular user-defined language.

Use $\tt USER_LEXER$ to plug in your own language-specific lexing solution. This enables you to define lexers for languages that are not supported by Oracle Text. It also enables you to define



a new lexer for a language that is supported but whose lexer is inappropriate for your application.

This section contains the following topics.

- USER_LEXER Routines
- USER_LEXER Limitations
- USER LEXER Attributes
- INDEX PROCEDURE
- INPUT_TYPE
- QUERY_PROCEDURE
- · Encoding Tokens as XML
- XML Schema for No-Location_ User-defined Indexing Procedure
- XML Schema for User-defined Indexing Procedure with Location
- XML Schema for User-defined Lexer Query Procedure

2.5.9.1 USER_LEXER Routines

The user-defined lexer you register with Oracle Text is composed of two routines that you must supply:

Table 2-28 User-Defined Routines for USER_LEXER

User-Defined Routine	Description
Indexing Procedure	Stored procedure (PL/SQL) which implements the tokenization of documents and stop words. Output must be an XML document as specified in this section.
Query Procedure	Stored procedure (PL/SQL) which implements the tokenization of query words. Output must be an XML document as specified in this section.

2.5.9.2 USER_LEXER Limitations

The following features are not supported with the USER LEXER:

- CTX DOC.GIST and CTX DOC.THEMES
- CTX_QUERY.HFEEDBACK
- ABOUT query operator
- CTXRULE index type
- VGRAM indexing algorithm

2.5.9.3 USER_LEXER Attributes

USER LEXER has the following attributes:



Table 2-29 USER_LEXER Attributes

Attribute	Attribute Value	
INDEX_PROCEDURE	Name of a stored procedure. No default provided.	
INPUT_TYPE	VARCHAR2, CLOB. Default is CLOB.	
QUERY_PROCEDURE	Name of a stored procedure. No default provided.	

2.5.9.4 INDEX PROCEDURE

This callback stored procedure is called by Oracle Text as needed to tokenize a document or a stop word found in the stoplist object.

Requirements

This procedure can be a PL/SQL stored procedure.

The index owner must have EXECUTE privilege on this stored procedure.

This stored procedure must not be replaced or dropped after the index is created. You can replace or drop this stored procedure after the index is dropped.

Parameters

Two different interfaces are supported for the user-defined lexer indexing procedure:

- VARCHAR2 Interface
- CLOB Interface

Restrictions

This procedure must not perform any of the following operations:

- Rollback
- Explicitly or implicitly commit the current transaction
- Enter any other transaction control statement
- Alter the session language or territory

The child elements of the root element tokens of the XML document returned must be in the same order as the tokens occur in the document or stop word being tokenized.

The behavior of this stored procedure must be deterministic with respect to all parameters.

2.5.9.5 INPUT_TYPE

Two different interfaces are supported for the User-defined lexer indexing procedure. One interface enables the document or stop word and the corresponding tokens encoded as XML to be passed as VARCHAR2 datatype whereas the other interface uses the CLOB datatype. This attribute indicates the interface implemented by the stored procedure specified by the INDEX PROCEDURE attribute.

- VARCHAR2 Interface
- CLOB Interface



2.5.9.5.1 VARCHAR2 Interface

Table 2-30 describes the interface that enables the document or stop word from stoplist object to be tokenized to be passed as VARCHAR2 from Oracle Text to the stored procedure and for the tokens to be passed as VARCHAR2 as well from the stored procedure back to Oracle Text.

Your user-defined lexer indexing procedure should use this interface when all documents in the column to be indexed are smaller than or equal to 32512 bytes and the tokens can be represented by less than or equal to 32512 bytes. In this case the CLOB interface given in Table 2-31 can also be used, although the VARCHAR2 interface will generally perform faster than the CLOB interface.

This procedure must be defined with the following parameters:

Table 2-30 VARCHAR2 Interface for INDEX_PROCEDURES

Parameter Position	Parameter Mode	Parameter Datatype	Description
1	IN	VARCHAR2	Document or stop word from stoplist object to be tokenized.
			If the document is larger than 32512 bytes then Oracle Text will report a document level indexing error.
2	IN OUT	VARCHAR2	Tokens encoded as XML.
			If the document contains no tokens, then either NULL must be returned or the tokens element in the XML document returned must contain no child elements.
			Byte length of the data must be less than or equal to 32512.
			To improve performance, use the NOCOPY hint when declaring this parameter. This passes the data by reference, rather than passing data by value.
			The XML document returned by this procedure should not include unnecessary whitespace characters (typically used to improve readability). This reduces the size of the XML document which in turn minimizes the transfer time.
			To improve performance, index_procedure should not validate the XML document with the corresponding XML schema at run-time.
			Note that this parameter is ${\tt IN}~{\tt OUT}$ for performance purposes. The stored procedure has no need to use the ${\tt IN}$ value.
3	IN	BOOLEAN	Oracle Text sets this parameter to TRUE when Oracle Text needs the character offset and character length of the tokens as found in the document being tokenized.
			Oracle Text sets this parameter to FALSE when Text is not interested in the character offset and character length of the tokens as found in the document being tokenized. This implies that the XML attributes off and len must not be used.

2.5.9.5.2 CLOB Interface

Table 2-31 describes the CLOB interface that enables the document or stop word from stoplist object to be tokenized to be passed as CLOB from Oracle Text to the stored procedure and for the tokens to be passed as CLOB as well from the stored procedure back to Oracle Text.

The user-defined lexer indexing procedure should use this interface when at least one of the documents in the column to be indexed is larger than 32512 bytes or the corresponding tokens are represented by more than 32512 bytes.



Table 2-31 CLOB Interface for INDEX_PROCEDURE

Parameter Position	Parameter Mode	Parameter Datatype	Description
1	IN	CLOB	Document or stop <i>word</i> from stoplist object to be tokenized.
2	IN OUT	CLOB	Tokens encoded as XML.
			If the document contains no tokens, then either NULL must be returned or the tokens element in the XML document returned must contain no child elements.
			To improve performance, use the NOCOPY hint when declaring this parameter. This passes the data by reference, rather than passing data by value.
			The XML document returned by this procedure should not include unnecessary whitespace characters (typically used to improve readability). This reduces the size of the XML document which in turn minimizes the transfer time.
			To improve performance, index_procedure should not validate the XML document with the corresponding XML schema at run-time.
			Note that this parameter is IN OUT for performance purposes. The stored procedure has no need to use the IN value. The IN value will always be a truncated CLOB.
3	IN	BOOLEAN	Oracle Text sets this parameter to TRUE when Oracle Text needs the character offset and character length of the tokens as found in the document being tokenized.
			Oracle Text sets this parameter to FALSE when Text is not interested in the character offset and character length of the tokens as found in the document being tokenized. This implies that the XML attributes off and len must not be used.

The first and second parameters are temporary CLOBS. Avoid assigning these CLOB locators to other locator variables. Assigning the formal parameter CLOB locator to another locator variable causes a new copy of the temporary CLOB to be created resulting in a performance hit.

2.5.9.6 QUERY_PROCEDURE

This callback stored procedure is called by Oracle Text as needed to tokenize *words* in the query. A space-delimited group of characters (excluding the query operators) in the query will be identified by Oracle Text as a *word*.

Requirements

This procedure can be a PL/SQL stored procedure.

The index owner must have EXECUTE privilege on this stored procedure.

This stored procedure must not be replaced or be dropped after the index is created. You can replace or drop this stored procedure after the index is dropped.

Restrictions

This procedure must not perform any of the following operations:



- Rollback
- Explicitly or implicitly commit the current transaction
- Enter any other transaction control statement
- Alter the session language or territory

The child elements of the root element tokens of the XML document returned must be in the same order as the tokens occur in the query *word* being tokenized.

The behavior of this stored procedure must be deterministic with respect to all parameters.

Parameters

Table 2-32 describes the interface for the user-defined lexer query procedure:

Table 2-32 User-defined Lexer Query Procedure XML Schema Attributes

Parameter Position	Parameter Mode	Parameter Datatype	Description
1	IN	VARCHAR2	Query word to be tokenized.
2	IN	CTX_ULEXER.WILDCARD_TAB	Character offsets of wildcard characters (% and _) in the query word. If the query word passed in by Oracle Text does not contain any wildcard characters then this index-by table will be empty.
			The wildcard characters in the query word must be preserved in the tokens returned in order for the wildcard query feature to work properly.
			The character offset is 0 (zero) based. Offset information follows USC-2 codepoint semantics.
3	IN OUT	VARCHAR2	Tokens encoded as XML.
			If the query word contains no tokens then either NULL must be returned or the tokens element in the XML document returned must contain no child elements.
			The length of the data must be less-than or equal to 32512 bytes.

2.5.9.7 Encoding Tokens as XML

The sequence of tokens returned by your stored procedure must be represented as an XML 1.0 document. The XML document must be valid with respect to the XML Schemas given in the following sections.

- XML Schema for No-Location_ User-defined Indexing Procedure
- XML Schema for User-defined Indexing Procedure with Location
- XML Schema for User-defined Lexer Query Procedure

Limitations

To boost performance of this feature, the XML parser in Oracle Text will not perform validation and will not be a full-featured XML compliant parser. This implies that only minimal XML features will be supported. The following XML features are not supported:



- Document Type Declaration (for example, <!DOCTYPE [...]>) and therefore entity
 declarations. Only the following built-in entities can be referenced: It, gt, amp, quot, and
 apos.
- CDATA sections.
- Comments.
- Processing Instructions.
- XML declaration (for example, <?xml version="1.0" ...?>).
- Namespaces.
- Use of elements and attributes other than those defined by the corresponding XML Schema.
- Character references (for example ট).
- xml:space attribute.
- xml:lang attribute

2.5.9.8 XML Schema for No-Location, User-defined Indexing Procedure

This section describes additional constraints imposed on the XML document returned by the user-defined lexer indexing procedure when the third parameter is FALSE. The XML document returned must be valid with respect to the following XML Schema:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="tokens">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:choice minOccurs="0" maxOccurs="unbounded">
          <xsd:element name="eos" type="EmptyTokenType"/>
          <xsd:element name="eop" type="EmptyTokenType"/>
          <xsd:element name="num" type="xsd:token"/>
          <xsd:group ref="IndexCompositeGroup"/>
        </xsd:choice>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
 Enforce constraint that compMem element must be preceded by word element
 or compMem element for indexing
  <xsd:group name="IndexCompositeGroup">
    <xsd:sequence>
      <xsd:element name="word" type="xsd:token"/>
     <xsd:element name="compMem" type="xsd:token" minOccurs="0"</pre>
maxOccurs="unbounded"/>
   </xsd:sequence>
 </xsd:group>
 <!-- EmptyTokenType defines an empty element without attributes -->
 <xsd:complexType name="EmptyTokenType"/>
</xsd:schema>
```

Here are some of the constraints imposed by this XML Schema:

The root element is tokens. This is mandatory. It has no attributes.

- The root element can have zero or more child elements. The child elements can be one of the following elements: eos, eop, num, word, and compMem. Each of these represent a specific type of token.
- The compMem element must be preceded by a word element or a compMem element.
- The eos and eop elements have no attributes and must be empty elements.
- The num, word, and compMem elements have no attributes. Oracle Text will normalize the
 content of these elements as follows: convert whitespace characters to space characters,
 collapse adjacent space characters to a single space character, remove leading and
 trailing spaces, perform entity reference replacement, and truncate to 255 bytes.

Table 2-33 describes the element names defined in the preceding XML Schema.

Table 2-33 User-defined Lexer Indexing Procedure XML Schema Element Names

Element	Description
word	This element represents a simple word token. The content of the element is the word itself. Oracle Text does the work of identifying this token as being a stop word or non-stop word and processing it appropriately.
num	This element represents an arithmetic number token. The content of the element is the arithmetic number itself. Oracle Text treats this token as a stop word if the stoplist preference has NUMBERS added as the stopclass. Otherwise this token is treated the same way as the word token.
	Supporting this token type is optional. Without support for this token type, adding the NUMERBS stopclass will have no effect.
eos	This element represents end-of-sentence token. Oracle Text uses this information so that it can support WITHIN SENTENCE queries.
	Supporting this token type is optional. Without support for this token type, queries against the SENTENCE section will not work as expected.
еор	This element represents end-of-paragraph token. Oracle Text uses this information so that it can support WITHIN PARAGRAPH queries.
	Supporting this token type is optional. Without support for this token type, queries against the PARAGRAPH section will not work as expected.
compMem	Same as the word element, except that the implicit word offset is the same as the previous word token.
	Support for this token type is optional.

Examples

Document: Vom Nordhauptbahnhof und aus der Innenstadt zum Messegelände.

Tokens:

```
<tokens>
    <word> VOM </word>
    <word> NORDHAUPTBAHNHOF </word>
    <compMem>NORD</compMem>
    <compMem>HAUPT </compMem>
    <compMem>BAHNHOF </compMem>
    <compMem>HAUPTBAHNHOF </compMem>
    <word> UND </word>
    <word> AUS </word>
    <word> DER </word>
    <word> INNENSTADT </word>
    <word> ZUM </word>
    <word> MESSEGELÄNDE </word>
```



```
<eos/></tokens>
```

Document: Oracle Database 11g Release 1

Tokens:

```
<tokens>
  <word> ORACLE11G</word>
  <word> RELEASE </word>
  <num> 1 </num>
</tokens>
```

Document: WHERE salary<25000.00 AND job = 'F&B Manager'

Tokens:

```
<tokens>
  <word> WHERE </word>
  <word> salary&lt;2500.00 </word>
  <word> AND </word>
  <word> job </word>
  <word> F&amp;B </word>
  <word> Manager </word>
  </tokens>
```

2.5.9.9 XML Schema for User-defined Indexing Procedure with Location

This section describes additional constraints imposed on the XML document returned by the user-defined lexer indexing procedure when the third parameter is TRUE. The XML document returned must be valid according to the following XML schema:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="tokens">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:choice minOccurs="0" maxOccurs="unbounded">
          <xsd:element name="eos" type="EmptyTokenType"/>
          <xsd:element name="eop" type="EmptyTokenType"/>
          <xsd:element name="num" type="DocServiceTokenType"/>
          <xsd:group ref="DocServiceCompositeGroup"/>
        </xsd:choice>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
 Enforce constraint that compMem element must be preceded by word element
 or compMem element for document service
 <xsd:group name="DocServiceCompositeGroup">
   <xsd:sequence>
     <xsd:element name="word" type="DocServiceTokenType"/>
     <xsd:element name="compMem" type="DocServiceTokenType" minOccurs="0"</pre>
          maxOccurs="unbounded"/>
    </xsd:sequence>
 </xsd:group>
 <!-- EmptyTokenType defines an empty element without attributes -->
  <xsd:complexType name="EmptyTokenType"/>
```



Some of the constraints imposed by this XML Schema are as follows:

- The root element is tokens. This is mandatory. It has no attributes.
- The root element can have zero or more child elements. The child elements can be one of the following elements: eos, eop, num, word, and compMem. Each of these represent a specific type of token.
- The compMem element must be preceded by a word element or a compMem element.
- The eos and eop elements have no attributes and must be empty elements.
- The num, word, and compMem elements have two mandatory attributes: off and len. Oracle
 Text will normalize the content of these elements as follows: convert whitespace characters
 to space characters, collapse adjacent space characters to a single space character,
 remove leading and trailing spaces, perform entity reference replacement, and truncate to
 255 bytes.
- The off attribute value must be an integer between 0 and 2147483647 inclusive.
- The len attribute value must be an integer between 0 and 65535 inclusive.

Table 2-33 describes the element types defined in the preceding XML Schema.

Table 2-34 describes the attributes defined in the preceding XML Schema.

Table 2-34 User-defined Lexer Indexing Procedure XML Schema Attributes

Attribute	Description
off	This attribute represents the character offset of the token as it appears in the document being tokenized.
	The offset is with respect to the character document passed to the user-defined lexer indexing procedure, not the document fetched by the datastore. The document fetched by the datastore may be pre-processed by the filter object or the section group object, or both, before being passed to the user-defined lexer indexing procedure.
	The offset of the first character in the document being tokenized is 0 (zero). Offset information follows USC-2 codepoint semantics.



Table 2-34 (Cont.) User-defined Lexer Indexing Procedure XML Schema Attributes

Attribute	Description
len	This attribute represents the character length (same semantics as SQL function LENGTH) of the token as it appears in the document being tokenized.
	The length is with respect to the character document passed to the user-defined lexer indexing procedure, not the document fetched by the datastore. The document fetched by the datastore may be pre-processed by the filter object or the section group object before being passed to the user-defined lexer indexing procedure.
	Length information follows USC-2 codepoint semantics.

Sum of off attribute value and len attribute value must be less than or equal to the total number of characters in the document being tokenized. This is to ensure that the document offset and characters being referenced are within the document boundary.

Example

Document: User-defined Lexer.

Tokens:

```
<tokens>
  <word off="0" len="4"> USE </word>
  <word off="5" len="7"> DEF </word>
  <word off="13" len="5"> LEX </word>
  <eos/>
  </tokens>
```

2.5.9.10 XML Schema for User-defined Lexer Query Procedure

This section describes additional constraints imposed on the XML document returned by the user-defined lexer query procedure. The XML document returned must be valid with respect to the following XML Schema:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="tokens">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:choice minOccurs="0" maxOccurs="unbounded">
          <xsd:element name="num" type="QueryTokenType"/>
          <xsd:group ref="QueryCompositeGroup"/>
        </xsd:choice>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
Enforce constraint that compMem element must be preceded by word element
or compMem element for query
-->
  <xsd:group name="QueryCompositeGroup">
    <xsd:sequence>
      <xsd:element name="word" type="QueryTokenType"/>
      <xsd:element name="compMem" type="QueryTokenType" minOccurs="0"</pre>
                                               maxOccurs="unbounded"/>
    </xsd:sequence>
```

```
</xsd:group>
 QueryTokenType defines an element with content and with an optional attribute
  <xsd:complexType name="QueryTokenType">
    <xsd:simpleContent>
      <xsd:extension base="xsd:token">
        <xsd:attribute name="wildcard" type="WildcardType" use="optional"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
 <xsd:simpleType name="WildcardType">
    <xsd:restriction base="WildcardBaseType">
      <xsd:minLength value="1"/>
      <xsd:maxLength value="64"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="WildcardBaseType">
    <xsd:list>
      <xsd:simpleType>
       <xsd:restriction base="xsd:unsignedShort">
          <xsd:maxInclusive value="378"/>
       </xsd:restriction>
     </xsd:simpleType>
    </xsd:list>
 </xsd:simpleType>
</xsd:schema>
```

Here are some of the constraints imposed by this XML Schema:

- The root element is tokens. This is mandatory. It has no attributes.
- The root element can have zero or more child elements. The child elements can be one of the following elements: num and word. Each of these represent a specific type of token.
- The compMem element must be preceded by a word element or a compMem element.

The purpose of <code>compMem</code> is to enable <code>USER_LEXER</code> queries to return multiple forms for a single query. For example, if a user-defined lexer indexes the word bank as <code>BANK(FINANCIAL)</code> and <code>BANK(RIVER)</code>, the query procedure can return the first term as a word and the second as a <code>compMem</code> element:

```
<tokens>
  <word>BANK(RIVER)</word>
  <compMem>BANK(FINANCIAL)</compMem>
</tokens>
```

See Table 2-35, "Table 2-35" for more on the compMem element.

- The num and word elements have a single optional attribute: wildcard. Oracle Text will
 normalize the content of these elements as follows: convert whitespace characters to
 space characters, collapse adjacent space characters to a single space character, remove
 leading and trailing spaces, perform entity reference replacement, and truncate to 255
 bytes.
- The wildcard attribute value is a white-space separated list of integers. The minimum number of integers is 1 and the maximum number of integers is 64. The value of the integers must be between 0 and 378 inclusive. The intriguers in the list can be in any order.

Table 2-33 describes the element types defined in the preceding XML Schema.

Table 2-35 describes the attribute defined in the preceding XML Schema.

Table 2-35 User-defined Lexer Query Procedure XML Schema Attributes

Attribute	Description
compMem	Same as the word element, but its implicit word offset is the same as the previous word token. Oracle Text will equate this token with the previous word token and with subsequent compMem tokens using the query EQUIV operator.
wildcard	Any % or _ characters in the query which are not escaped by the user are considered wildcard characters because they are replaced by other characters. These wildcard characters in the query must be preserved during tokenization in order for the wildcard query feature to work properly. This attribute represents the character offsets (same semantics as SQL function LENGTH) of wildcard characters in the content of the element. Oracle Text will adjust these offsets for any normalization performed on the content of the element. The characters pointed to by the offsets must either be % or _ characters.
	The offset of the first character in the content of the element is 0. Offset information follows USC-2 codepoint semantics.
	If the token does not contain any wildcard characters then this attribute must not be specified.

Examples

Query word: pseudo-%morph%

Tokens:

```
<tokens>
  <word> PSEUDO </word>
  <word wildcard="1 7"> %MORPH% </word>
</tokens>
```

Query word: <%>

Tokens:

```
<tokens>
  <word wildcard="5"> &lt;%&gt; </word>
</tokens>
```

2.5.10 WORLD_LEXER

A simple lexer that can index documents in any language or mixed languages. Works with short strings and long documents. Does not support stemming or other lexer-related attributes.

Use the <code>WORLD_LEXER</code> to index text columns that contain documents of different languages. For example, use this lexer to index a text column that stores English, Japanese, and German documents.

WORLD_LEXER differs from MULTI_LEXER in that WORLD_LEXER automatically detects the language(s) of a document. Unlike MULTI_LEXER, WORLD_LEXER does not require you to have a language column in your base table nor to specify the language column when you create the index. Moreover, it is not necessary to use sub-lexers, as with MULTI_LEXER. (See "MULTI_LEXER".)



WORLD_LEXER supports all database character sets, and for languages whose character sets are Unicode-based, it supports the Unicode 5.0 standard. For a list of languages that WORLD_LEXER can work with, see "World Lexer Features".

The WORLD LEXER has the following attributes:

Table 2-36 WORLD_LEXER Attributes

Attribute	Attribute Value
mixed_case	Enables mixed-case (upper- and lower-case) searches of text (for example, <i>cat</i> and <i>Cat</i>). Allowable values are YES and NO (default).
printjoins	Specify the non alphanumeric characters that, when they appear anywhere in a word (beginning, middle, or end), are processed as alphanumeric and included with the token in the Text index. This includes printjoins that occur consecutively. See Basic Lexer "printjoins".
skipjoins	Specify the non-alphanumeric characters that, when they appear within a word, identify the word as a single token; however, the characters are not stored with the token in the Text index. See Basic Lexer "skipjoins".

Rules for PRINTJOIN and SKIPJOIN Characters

Refer to "Rules for PRINTJOIN and SKIPJOIN Characters" in JAPANESE_VGRAM_LEXER.

WORLD_LEXER Example

The following is an example of creating an index using WORLD LEXER.

2.6 Wordlist Type

Use the wordlist preference to enable the query options such as stemming, fuzzy matching for your language. You can also use the wordlist preference to enable substring and prefix indexing, which improves performance for wildcard queries with CONTAINS and CATSEARCH.

To create a wordlist preference, you must use <code>BASIC_WORDLIST</code>, which is the only type available.

- BASIC_WORDLIST
- BASIC_WORDLIST Example

2.6.1 BASIC_WORDLIST

Use BASIC_WORDLIST to enable stemming and fuzzy matching or to create prefix indexes with Text indexes.

Table 2-37 BASIC_WORDLIST Attributes

Attribute Attribute Values

stemmer

Specify which language stemmer to use. You can specify one of the following stemmers:

- NULL (no stemming)
- AUTO (Automatic language-detection for stemming, derived from the database session language. For example, if the database session language is American or English, then the English stemmer is used. Note that the STEMMER=AUTO attribute value resolves the environment language (NLS_LANG) to the supported languages. Does not autodetect Japanese.)
- Afrikaans
- Arabic
- Basque
- Belarusian
- Bokmal (Norwegian)
- Bulgarian
- Catalan
- Croatian
- Czech
- Danish
- Derivational (English derivational)
- Dutch
- English (English inflectional)
- Estonian
- Finnish
- French
- Galician
- German
- Greek
- Hebrew
- Hindi
- Hungarian
- Icelandic
- Indonesian
- Italian
- Japanese
- Latvian
- Lithuanian
- Macedonian
- Malay
- Nynorsk (Norwegian)
- Persian (Farsi)
- Polish
- Portuguese
- Romanian
- Russian
- Serbian
- Slovak



Table 2-37 (Cont.) BASIC_WORDLIST Attributes

Attribute	Attribute Values	
	• Slovenian	
	• Spanish	
	• Swedish	
	• Turkish	
	• Ukrainian	
	• Urdu	
fuzzy_match	Specify which fuzzy matching cluster to use. You can specify one of the following types:	
	AUTO (Automatic language detection for stemming)	
	CHINESE_VGRAM	
	Dutch	
	English	
	French	
	GENERIC	
	German	
	Italian	
	JAPANESE_VGRAM	
	Korean	
	OCR	
	Spanish	
fuzzy_score	Specify a default lower limit of fuzzy score. Specify a number between 1 and 80. Text with scores below this number is not returned. Default is 60.	
fuzzy_numresults	Specify the maximum number of fuzzy expansions. Use a number between 0 and 5,000. Default is 100.	
substring_index	Specify TRUE for Oracle Text to create a substring index. A substring index improves left-truncated and double-truncated wildcard queries such as <i>%ing</i> or <i>%benz%</i> . Default is FALSE.	
prefix_index	Specify TRUE to enable prefix indexing. Prefix indexing improves performance for right truncated wildcard searches such as TO%. Default is ${\tt FALSE}$.	
prefix_min_length	Specify the minimum length of indexed prefixes. Default is 1. Length information must follow USC-2 codepoint semantics.	
prefix_max_length	Specify the maximum length of indexed prefixes. Default is 64. Length information must follow USC-2 codepoint semantics.	
wildcard_maxterms	Specify the maximum number of terms in a wildcard expansion. The maximum value is 50000 and the default value is 20000 . If you specify a value of 0 , then the number of wildcard expansions will be unbounded. Note that when set to 0 , the system may run out of memory due to the high number of wildcard expansions.	
ndata_base_letter	Specify whether characters that have diacritical marks are converted to their base form before being stored in the Text index or queried by the NDATA operator.	
	FALSE (default) or TRUE	



Table 2-37 (Cont.) BASIC_WORDLIST Attributes

Attribute	Attribute Values
ndata_alternate_spelling	Specify whether to enable alternate spelling for German, Danish, and Swedish. Enabling alternate spelling allows you to index NDATA section data and query using the NDATA operator in alternate form.
	FALSE (default) or TRUE
	When set to FALSE, no alternate spelling is used.
ndata_thesaurus	Name of the thesaurus used for alternate name expansion.
ndata_join_particles	A list of colon-separated name particles that can be joined with a name that follows them.
reverse_index	Specify whether to enable the creation of another index on \$I to provide better performance for left truncated queries. These are queries where one or more tokens have a leading wildcard and no trailing wildcard, for example, the %racle %atabase.
	When set to TRUE, it creates a new index \$V on \$I on reverse (token_text). Default is FALSE.
wildcard_index	Specify TRUE to enable wildcard indexing. Wildcard indexing supports fast and efficient wildcard search for all wildcard expressions. The default value is FALSE.
wildcard_index_k	Specify the size of grams for the K-gram index. The value can range between 2 and 5 and the default value is 3.

stemmer

Specify the stemmer used for word stemming in Text queries. When you do not specify a value for STEMMER, the default is ENGLISH.

Specify AUTO for the system to automatically set the stemming language according to the language setting of the database session. If the database language is American or English, then the ENGLISH stemmer is automatically used. Otherwise, the stemmer that maps to the database session language is used.

When there is no stemmer for a language, the default is NULL. With the NULL stemmer, the stem operator is ignored in queries.

You can create your own stemming user-dictionary.



The STEMMER attribute of BASIC_WORDLIST preference is ignored if the INDEX_STEMS attribute of the AUTO_LEXER preference is set to YES. In this case, the same stemmer that is used by AUTO_LEXER during indexing is used to determine the stem of the query term during query.

fuzzy_match

Sspecify which fuzzy matching routines are used for the column. Fuzzy matching is currently supported for English, Japanese, and, to a lesser extent, the Western European languages.



Note:

The fuzzy_match attributes value for Chinese and Korean are dummy attribute values that prevent the English and Japanese fuzzy matching routines from being used on Chinese and Korean text.

The default for fuzzy match is GENERIC.

Specify AUTO for the system to automatically set the fuzzy matching language according to language setting of the session.

fuzzy_score

Specify a default lower limit of fuzzy score. Specify a number between 1 and 80. Text with scores below this number are not returned. The default is 60.

Fuzzy score is a measure of how close the expanded word is to the query word. The higher the score the better the match. Use this parameter to limit fuzzy expansions to the best matches.

fuzzy_numresults

Specify the maximum number of fuzzy expansions. Use a number between 0 and 5000. The default is 100.

Setting a fuzzy expansion limits the expansion to a specified number of the best matching words.

substring_index

Specify TRUE for Oracle Text to create a substring index. A substring index improves performance for left-truncated or double-truncated wildcard queries such as %ing or %benz%. The default is false.

Limitations of substring_index:

Oracle recommends using the wildcard_index attribute over substring_index. See wildcard_index. Substring indexing has the following impact on indexing and disk resources:

- Index creation and DML processing is up to 4 times slower.
- Index creation with substring_index enabled requires more rollback segments during index flushes than with substring_index off. Do either of the following when creating a substring index:
 - Make available double the usual rollback.
 - Decrease the index memory to reduce the size of the index flushes to disk.

prefix index

Specify yes to enable prefix indexing. Prefix indexing improves performance for right truncated wildcard searches such as TO%. Default is NO.



Enabling prefix indexing increases index size.

Prefix indexing chops up tokens into multiple prefixes to store in the \$I table. For example, words TOKEN and TOY are normally indexed as follows in the \$I table:



Token	Туре	Information
TOKEN	0	DOCID 1 POS 1
TOY	0	DOCID 1 POS 3

With prefix indexing, Oracle Text indexes the prefix substrings of these tokens as follows with a new token type of 6:

Token	Туре	Information
TOKEN	0	DOCID 1 POS 1
TOY	0	DOCID 1 POS 3
Т	6	DOCID 1 POS 1 POS 3
TO	6	DOCID 1 POS 1 POS 3
TOK	6	DOCID 1 POS 1
TOKE	6	DOCID 1 POS 1
TOKEN	6	DOCID 1 POS 1
TOY	6	DOCID 1 POS 3

Wildcard searches such as TO% are now faster because Oracle Text does no expansion of terms and merging of result sets. To obtain the result, Oracle Text need only examine the (TO,6) row.

prefix_min_length

Specify the minimum length of indexed prefixes. Default is 1.

For example, setting prefix_min_length to 3 and prefix_max_length to 5 indexes all prefixes between 3 and 5 characters long.



A wildcard search whose pattern is below the minimum length or above the maximum length is searched using the slower method of equivalence expansion and merging.

prefix_max_length

Specify the maximum length of indexed prefixes. Default is 64.

For example, setting prefix_min_length to 3 and prefix_max_length to 5 indexes all prefixes between 3 and 5 characters long.



A wildcard search whose pattern is below the minimum length or above the maximum length is searched using the slower method of equivalence expansion and merging.



wildcard maxterms

Specify the maximum number of terms in a wildcard (%) expansion. Use this parameter to keep wildcard query performance within an acceptable limit. When the wildcard query expansion exceeds this number, Oracle Text returns the following error:

```
ORA-29902: error in executing ODCIIndexStart() routine ORA-20000: Oracle Text error: DRG-51030: wildcard query expansion resulted in too many terms
```

In such cases, use a more restrictive query so that it results in fewer matches or increase the value of wildcard maxterms. You can also set wildcard maxterms to 0 to ignore the limit.



If the value of wildcard_maxterms is set as 0, the query might fail and returns the above error again if too many terms are matched by the wildcard search term.

You can also capture the above error and display your own less terse message.

Note:

Search terms with wildcard queries having only the wildcard character, for example: \$, $\$_-\$$, and $\$_-$, are threaded as stopwords.

Note:

wildcard_maxterms is independent of the new WILDCARD_INDEX option. wildcard_maxterms can be set even if WILDCARD_INDEX is not used.

ndata base letter

Specify whether characters that have diacritical marks (umlauts, cedillas, acute accents, and so on) are converted to their base form before being stored in the Text index or queried by the NDATA operator. The default is FALSE (base-letter conversion disabled).

ndata_alternate_spelling

Specify whether to enable alternate spelling for German, Danish, and Swedish. Enabling alternate spelling allows you to index \mathtt{NDATA} section data and query using the \mathtt{NDATA} operator in alternate form.

When ndata_base_letter is enabled at the same time as ndata_alternate_spelling, NDATA section data is serially transformed first by alternate spelling and then by base lettering.

ndata_thesaurus

Specify a name of the thesaurus used for alternate name expansion. The indexing engine expands names in documents using synonym rings in the thesaurus. A user should make use of homographic disambiguating feature of the thesaurus to distinguish common nicknames. An example is:



```
Albert
SYN Al
SYN Bert
Alfred
SYN Al
SYN Fred
```

A simple definition such as the above will put Albert, Alfred, Al, Bert, and Fred into the same synonym ring. This will cause an unexpected expansion such that the expansion of Bert includes Fred. To prevent this, you can use homographic disambiguation as in:

```
Albert
SYN Al (Albert)
SYN Bert (Albert)
Alfred
SYN Al (Alfred)
SYN Fred (Alfred)
```

This forms two synonym rings, Albert-Al-Bert and Alfred-Al-Fred. Thus, the expansion of Bert no longer includes Fred. A more detailed example is:

```
begin
   ctx_ddl.create_preference('NDAT_PREF', 'BASIC_WORDLIST');
   ctx_ddl.set_attribute('NDATA_PREF', 'NDATA_ALTERNATE_SPELLING', 'FALSE');
   ctx_ddl.set_attribute('NDATA_PREF', 'NDATA_BASE_LETTER', 'TRUE');
   ctx_ddl.set_attribute('NDATA_PREF', 'NDATA_THESAURUS', 'NICKNAMES');
end;
```

Note:

A sample thesaurus for names can be found in the <code>\$ORACLE_HOME/ctx/sample/thes</code> directory. This file is <code>dr0thsnames.txt</code>.

ndata join particles

Specify a list of colon-separated name particles that can be joined with a name that follows them. A name particle, such as da, is written separately from or joined with its following name like da Vinci or daVinci. The indexing engine generates index data for both separated and join versions of a name when it finds a name particle specified in this preference. The same happens in the query processing for better recall.

reverse index

Reverse index allows for fast searches on left-truncated search terms.

Indexed words are stored in the token table (\$I) which has an index (\$X) on it. Normally, if a search term such as "%xxx" is used in a query, the \$X index cannot be used. So, a full table scan of the \$I table is necessary, which can lead to poor search performance.

Setting <code>REVERSE_INDEX</code> to <code>TRUE</code> creates an extra index (\$V) on a reverse form of the tokens. This allows for indexed lookups for left-truncated terms, leading to much better query performance for such terms.

REVERSE_INDEX speeds up searching of tokens with leading wildcards such as the second word in the search "oracle %base". If the token has both leading and trailing wildcards such as "oracle %bas%" this attribute will not help and the SUBSTRING_INDEX option should be used instead

Specify the attribute as a part of the wordlist preference and set it to TRUE or FALSE. Default is FALSE. Set this attribute using CTX_DDL.SET_ATTRIBUTE procedure or using ALTER INDEX REBUILD statement as used in any wordlist preference.

Syntax



```
ctx_ddl.set_attribute(worlist_pref_name, 'REVERSE_INDEX', BOOLEAN);
```

worlist_pref_name

Specify the first argument as the wordlist preference name.

REVERSE INDEX

Specify the wordlist preference name as REVERSE INDEX.

BOOLEAN

The attribute can be set to TRUE or FALSE. By default, the value is FALSE.

The following example creates a wordlist preference and sets REVERSE INDEX to TRUE:

```
exec ctx_ddl.create_preference('wrdlst', 'BASIC_WORDLIST');
exec ctx_ddl.set attribute('wrdlst', 'REVERSE INDEX', 'TRUE');
```

The following traces are added for the Reverse Index \$V\$ which can be used to track timing and usage of this index at guery time.

Trace ID	Trace Name	Description
37	TRACE_QRY_VV_TIME	Time spent in executing the \$V cursor
38	TRACE_QRY_VF_TIME	Time spent in fetching rows from \$V
39	TRACE_QRY_V_ROWS	Number of rows with \$V fetched metadata

wildcard_index

Wildcard indexing supports fast and efficient wildcard search for all wildcard expressions. It is set using CTX DDL.SET ATTRIBUTE procedure.

Setting the WILDCARD INDEX to TRUE enables wildcard indexing.

Syntax

```
ctx ddl.set attribute(<wordlist pref name>, 'WILDCARD INDEX', BOOLEAN);
```

wordlist_pref_name

Specify the first argument as the wordlist preference name.

WILDCARD INDEX

Specify the wordlist preference name as WILDCARD INDEX.

BOOLEAN

The attribute can be set to TRUE or FALSE.

The following example creates a wordlist preference and sets WILDCARD INDEX to TRUE:

```
begin
    ctx_ddl.create_preference('mywordlist','BASIC_WORDLIST');
    ctx_ddl.set_attribute('mywordlist','WILDCARD_INDEX','TRUE');
end:
```

Optimization of Wildcard Index

WILDCARD_INDEX can be optimized either as part of full optimize or as part of section type optimize.

The following two examples are ways of optimizing a wildcard index:





Wildcard indexing is supported for languages which only use single-byte characters.

wildcard index k

The WILDCARD_INDEX uses a technology known as K-grams (fixed-length substring particles). WILDCARD_INDEX_K defines the size of these grams (K). The value can range between 2 and 5. The default value is 3. Set this attribute using CTX_DDL.SET_ATTRIBUTE procedure or using ALTER INDEX REBUILD statement as used in any wordlist preference.

```
Note:

WILDCARD_INDEX must be set to TRUE before setting WILDCARD_INDEX_K.
```

The following are some considerations before changing the value of K from the default value of 3:

- Query terms that are shorter than the value of K cannot be retrieved using K-gram indexing.
- Decreasing the value of K increases the storage requirements and increasing the value of K decreases the storage requirements.
- Wildcard query terms must have at least K consecutive non-wildcard characters to use K-gram indexing. For example, if K value is 3, queries like "%abc%" or "%abcd%" can use K-gram indexing. For the same K value, queries like "%ab%" cannot use K-gram indexing.
- Wildcard query terms having at least K-1 consecutive non-wildcard characters at the beginning or end of the query term, can use K-gram indexing. For example, if K value is 3, queries like "ab%" and "%ab" can use k-gram indexing.

The following example creates a wordlist preference and enables K-gram indexing with a K value of 4:

```
begin
    ctx_ddl.create_preference('mywordlist','BASIC_WORDLIST');
    ctx_ddl.set_attribute('mywordlist','WILDCARD_INDEX','TRUE');
    ctx_ddl.set_attribute('mywordlist','WILDCARD_INDEX_K',4);
end;
```

Related Topics

- Oracle Text CONTAINS Query Operators
- Alternate Spelling



Stemming User-Dictionaries

You can create a user-dictionary for your own language to customize how words are decomposed.

• Base-Letter Conversion

2.6.2 BASIC_WORDLIST Example

The following example shows the use of the BASIC WORDLIST type.

- Enabling Fuzzy Matching and Stemming
- Enabling Sub-string and Prefix Indexing
- Setting Wildcard Expansion Limit

2.6.2.1 Enabling Fuzzy Matching and Stemming

The following example enables stemming and fuzzy matching for English. The preference STEM_FUZZY_PREF sets the number of expansions to the maximum allowed. This preference also instructs the system to create a substring index to improve the performance of double-truncated searches.

```
begin
  ctx_ddl.create_preference('STEM_FUZZY_PREF', 'BASIC_WORDLIST');
  ctx_ddl.set_attribute('STEM_FUZZY_PREF','FUZZY_MATCH','ENGLISH');
  ctx_ddl.set_attribute('STEM_FUZZY_PREF','FUZZY_SCORE','1');
  ctx_ddl.set_attribute('STEM_FUZZY_PREF','FUZZY_NUMRESULTS','5000');
  ctx_ddl.set_attribute('STEM_FUZZY_PREF','SUBSTRING_INDEX','TRUE');
  ctx_ddl.set_attribute('STEM_FUZZY_PREF','STEMMER','ENGLISH');
end;
```

To create the index in SQL, enter the following statement:

```
create index fuzzy_stem_subst_idx on mytable ( docs )
  indextype is ctxsys.context parameters ('Wordlist STEM FUZZY PREF');
```

2.6.2.2 Enabling Sub-string and Prefix Indexing

The following example sets the wordlist preference for prefix and sub-string indexing. For prefix indexing, it specifies that Oracle Text create token prefixes between 3 and 4 characters long:

```
begin

ctx_ddl.create_preference('mywordlist', 'BASIC_WORDLIST');
ctx_ddl.set_attribute('mywordlist','PREFIX_INDEX','TRUE');
ctx_ddl.set_attribute('mywordlist','PREFIX_MIN_LENGTH',3);
ctx_ddl.set_attribute('mywordlist','PREFIX_MAX_LENGTH', 4);
ctx_ddl.set_attribute('mywordlist','SUBSTRING_INDEX', 'YES');
end;
```

2.6.2.3 Setting Wildcard Expansion Limit

Use the wildcard_maxterms attribute to set the maximum allowed terms in a wildcard expansion.

```
--- create a sample table drop table quick; create table quick
```



```
quick id number primary key,
            varchar(80)
    text
  );
--- insert a row with 10 expansions for 'tire%'
insert into quick ( quick id, text )
  values ( 1, 'tire tirea tireb tirec tired tiree tiref tireg tireh tirei tirej');
--- create an index using wildcard maxterms=100
begin
    Ctx_Ddl.Create_Preference('wildcard_pref', 'BASIC_WORDLIST');
    ctx ddl.set attribute('wildcard pref', 'wildcard maxterms', 100);
end;
create index wildcard idx on quick(text)
    indextype is ctxsys.context
    parameters ('Wordlist wildcard pref');
--- query on 'tire%' - should work fine
select quick id from quick
  where contains (text, 'tire%') > 0;
--- now re-create the index with wildcard maxterms=5
drop index wildcard idx ;
begin
    Ctx Ddl.Drop Preference('wildcard pref');
    Ctx Ddl.Create Preference('wildcard pref', 'BASIC WORDLIST');
    ctx_ddl.set_attribute('wildcard_pref', 'wildcard_maxterms', 5);
end;
create index wildcard idx on quick(text)
    indextype is ctxsys.context
    parameters ('Wordlist wildcard pref');
--- query on 'tire%' gives "wildcard query expansion resulted in too many terms"
select quick id from quick
  where contains (text, 'tire%') > 0;
```

2.7 Storage Types

Use the storage preference to specify tablespace and creation parameters for tables associated with a Text index. The system provides a single storage type called BASIC STORAGE:

Table 2-38 Storage Types

Туре	Description
BASIC_STORAGE	Indexing type used to specify the tablespace and creation parameters for the database tables and indexes that constitute a Text index.

2.7.1 BASIC_STORAGE

The BASIC_STORAGE indexing type specifies the tablespace and creation parameters for the database tables and indexes that constitute a Text index.

The clause you specify is added to the internal CREATE TABLE (CREATE INDEX for the <code>i_index_clause</code>) statement at index creation. You can specify most allowable clauses, such as storage, LOB storage, or partitioning. However, you cannot specify an index organized table clause.

You can store Text index tables in the In-Memory Column Store (IM column store) by specifying <code>inmemory</code> in the storage clause for that table. IM column store is supported for the types of tables represented by the following storage attributes: <code>I_TABLE_CLAUSE</code>, <code>R_TABLE_CLAUSE</code>, <code>G_TABLE_CLAUSE</code>, <code>O_TABLE_CLAUSE</code>, <code>D_TABLE_CLAUSE</code>, <code>SN_TABLE_CLAUSE</code>, and <code>E_TABLE_CLAUSE</code>.

This section contains the following topics.

- BASIC_STORAGE Attributes
- BASIC_STORAGE Default Behavior
- BASIC_STORAGE Examples

See Also:

- Oracle Database SQL Language Reference for more information about how to specify CREATE INDEX statement
- Oracle Database SQL Language Reference for more information about how to specify CREATE TABLE statement

2.7.1.1 BASIC_STORAGE Attributes

The BASIC STORAGE indexing type supports these attributes for database tables and indexes.



Table 2-39 BASIC_STORAGE Attributes

Attribute Attribute Value Parameter clause to improve the query performance for the CONTEXT index that is big io extensively used for IO operations. It uses SECUREFILES, and hence the tablespace must use automatic segment space management (ASSM). This clause mainly improves the query performance for rotational disks, where seeks are expensive compared to serial reads. Creating an index with the BIG IO index option requires the CREATE TRIGGER privilege, as a temporary trigger is created during the indexing process. There is not much of a query performance improvement when the data storage is on solid state disks. Set it to YES to enable the BIG $\,$ IO index option for the CONTEXT index. The default is NO. **Note**: BIG IO index option is not supported for local Oracle Text search index. The BIG IO attribute of the CONTEXT indextype is deprecated with Oracle Database 23ai, and can be disabled or removed in a future release. Oracle recommends that you allow this value to be set to its default value of N. BIG IO was introduced to reduce the cost of seeks when index postings exceeded 4KB in length. However, the internal code is relatively inefficient, and the attribute cannot be combined with newer index options. Seek cost is much less relevant for solid state disks or non-volatile memory devices (NVMe), and seek cost is irrelevant when postings are cached. This setting is therefore of little benefit for most indexes. Parameter clause to specify the storage clause for the DR\$ INDEX NAME\$C table. c table clause Specify the storage and tablespace clauses to add to the end of the internal CREATE INDEX statements. To understand the purpose of DR\$ INDEX NAME\$C, see Oracle Text Application Developer's Guide. d table clause Parameter clause to specify the storage clause for the \$D table. This clause may be specified if the forward index feature is being used. The forward index feature is used to increase the query performance while calculating snippets. If the d table clause is manually set, then it is recommended that you choose SecureFiles with high compression for the document blob column doc of the \$D table. If the d table clause is not set, then the document blob uses SecureFiles by default, if the index owner's default tablespace is ASSM and the database compatible parameter is 11.0 or higher. The \$D table is created to save a copy of a document into the index by either specifying a save copy column or by specifying the save copy storage attribute. forward index Parameter clause to improve the performance of the following CTX DOC package procedures: ctx doc.snippet ctx doc.highlight ctx doc.markup Set it to TRUE to enable the forward index feature. This creates the \$O table.

The \$O table stores the mapping information from the token offsets in the \$I table

to character offsets in the indexed documents.

The default is FALSE.



Table 2-39 (Cont.) BASIC_STORAGE Attributes

Attribute	Attribute Value
g_index_clause	Parameter clause for the \$H btree index on the \$G table.
	Specify the storage and tablespace clauses to add to the end of the internal CREATE INDEX statement.
	When a CONTEXT index is created with the STAGE_ITAB index option, an empty \$G table is created with the \$H btree index on it. Use the g_index_clause clause in conjunction with the STAGE_ITAB index option for improving the query performance for the CONTEXT index that is extensively used for DML operations.
g_table_clause	Parameter clause for the \$G table.
	Specify the storage and tablespace clauses to add to the end of the internal CREATE TABLE statement.
	When a CONTEXT index is created with the STAGE_ITAB index option, an empty \$G table is created with the \$H btree index on it. Use the g_table_clause clause in conjunction with the STAGE_ITAB index option for improving the query performance for the CONTEXT index that is extensively used for DML operations.
i_index_clause	Parameter clause for dr\$indexname\$X index creation. Specify storage and tablespace clauses to add to the end of the internal CREATE INDEX statement. The default clause is: 'COMPRESS 2', which instructs Oracle Text to compress this index table.
	If you choose to override the default, Oracle recommends including COMPRESS 2 in your parameter clause to compress this table, because such compression saves disk space and helps query performance.
i_rowid_index_clause	Parameter clause to specify the storage clause for the \$R index on dr\$rowid column of the \$I table. Specify storage and tablespace clauses to add to the end of the internal CREATE INDEX statement.
	This clause is only used by the CTXCAT index type.
	Note : The Oracle Text indextype CTXCAT is deprecated with Oracle Database 23ai. The indextype itself, and it's operator CTXCAT, can be removed in a future release.
	CTXCAT was introduced when indexes were typically a few megabytes in size. Modern, large indexes, can be difficult to manage with CTXCAT. The addition of index sets to CTXCAT can be achieved more effectively by the use of FILTER BY and ORDER BY columns, or SDATA, or both, in the CONTEXT indextype. CTXCAT is therefore rarely an appropriate choice. Oracle recommends that you choose the more efficient CONTEXT indextype.
i_table_clause	Parameter clause for dr\$indexname\$I table creation. Specify storage and tablespace clauses to add to the end of the internal CREATE TABLE statement.
	The \$I table is the index data table.
	Note: Oracle strongly recommends that you do not specify "disable storage in row" for \$I LOBs, as this greatly degrades the query performance.
k_table_clause	Parameter clause for dr\$indexname\$K table creation. Specify storage and tablespace clauses to add to the end of the internal CREATE TABLE statement.
	The K table is the keymap table.
kd_index_clause	Parameter clause for $\$$ KD table creation. Specify storage and tablespace clauses to add to the end of the internal CREATE TABLE statement.
	The RD table is a btree index on top of the R table. It facilitates a quick docid-to-rowid (KD) mapping. Docids are used internally by Oracle Text, and ROWIDs are used by the database.



Table 2-39 (Cont.) BASIC_STORAGE Attributes

Attribute	Attribute Value			
kr_index_clause	Parameter clause for \$KR table creation. Specify storage and tablespace clauses to add to the end of the internal CREATE TABLE statement.			
	Similar to the \$KD table, the \$KR table is a btree index on top of the \$K table. It facilitates a quick rowid-to-docid mapping (KR) mapping. Docids are used internally by Oracle Text and ROWIDs are used by the database.			
kg_table_clause	Parameter clause for \$KG table creation. Specify storage and tablespace clauses to add to the end of the internal CREATE TABLE statement.			
	The \$KG table stores the k-gram index to facilitate efficient wildcard search.			
kg_index_clause	Parameter clause for \$KGI index creation. Specify storage and tablespace clauses to add to the end of the internal CREATE INDEX statement.			
n_table_clause	Parameter clause for dr\$indexname\$N table creation. Specify storage and tablespace clauses to add to the end of the internal CREATE TABLE statement.			
	The \$N table is the negative list table which keeps track of deleted document IDs. These document IDs must be cleaned up by index optimization.			
o_table_clause	Parameter clause to specify the storage clause for the \$0 table.			
	This clause may be specified if the forward index feature is being used. The forward index feature is used to increase the query performance while calculating snippets.			
	If the o_table_clause is manually set, then it is recommended that you choose SecureFiles with high compression for the document blob column mapping of the \$O table. If the o_table_clause is not set, then the document blob uses SecureFiles by default, if the index owner's default tablespace is ASSM and the database compatible parameter is 11.0 or higher.			
	The \$0 table is created when the forward index feature is enabled by specifying the forward_index storage attribute. The \$0 table stores the mapping information from the token offsets in the \$1 table to character offsets in the indexe documents.			
p_table_clause	Parameter clause for the substring index if you have enabled SUBSTRING_INDEX in the BASIC WORDLIST.			
	Specify storage and tablespace clauses to add to the end of the internal CREATE INDEX statement. The \$P table is an index-organized table so the storage clause you specify must be appropriate to this type of table.			
q_table_clause	Parameter clause to specify the storage clause for the DR\$INDEX_NAME\$Q table. Specify the storage and tablespace clauses to add to the end of the internal CREATE INDEX statements.			
query_filter_cache_size	Parameter clause to specify the maximum size of the query filter cache in bytes. The query filter cache is allocated out of the shared pool, so its maximum size must be smaller than the shared pool size. When this storage preference is set at the partition level, it is implicitly set at the index level. The default is 0.			
	Note: Starting in Oracle Database Release 21c, CTXFILTERCACHE is deprecated and also CTX_FILTER_CACHE_STATISTICS and QUERY_FILTER_CACHE_SIZE.			



Table 2-39 (Cont.) BASIC_STORAGE Attributes

Attribute	Attribute Value
r_table_clause	Parameter clause for dr\$indexname\$R table creation. Specify storage and tablespace clauses to add to the end of the internal CREATE TABLE statement.
	The \$R table is the ROWID table.
	The default clause is: 'LOB (DATA) STORE AS (CACHE)'
	If you modify this attribute, always include this clause for good performance. Note: When you set the COMPATIBLE database parameter to 18.1 or higher, all Oracle Text indexes are created using the default FAST_DML option, that is, the indexes will not have the \$R mapping table.
s_table_clause	Parameter clause for dr\$indexname\$S table creation*. Specify storage and tablespace clauses to add to the end of the internal CREATE TABLE statement. The default clause is nocompress.
	* For performance reasons, \$S table must be created on a tablespace with db block size >= 4K without overflow segment and without a PCTTHRESHOLD clause. If \$S is created on a tablespace with db block size < 4K, or is created with an overflow segment or with PCTTHRESHOLD clause, then appropriate errors will be raised during CREATE INDEX.
	The S table is the table that stores SDATA section values.
	If this clause is specified for a storage preference in an index without SDATA, then it will have no effect on the index, and index creation will still succeed.
save_copy	Parameter clause to specify saving the document to the \$D index table.
	Specify this clause to use the forward index feature for increasing the query performance while calculating snippets.
	Set it to PLAINTEXT to save the copy of a document in the \$D table in the plaintext format. This improves the performance of snippet generation, since it does not invoke the datastore or filter to fetch the text. This also improves the performance of highlight.
	Set it to FILTERED to save the copy of a document in the \$D table in the filtered (HTML) format. This improves the performance of highlight and markup, but requires more disk space than plaintext format. It is less efficient for snippets generation, since the HTML markup must be removed during the creation of snippets.
	The default is ${\tt NONE}$, and the copy of a document is not saved in the ${\tt \$D}$ table.
save_copy_max_size	Parameter clause to specify the maximum size of a document to save in the \$D table using a basic_storage attribute.
	If the document size is greater than the size specified in this attribute, the truncated version of the document having the size specified in this attribute is saved in the \$D table.
	If the \$D table is using SecureFiles with compression for the document blob, then the <code>save_copy_max_size</code> restriction is applied on the document size before compression.
	The default is 0, and the whole document is saved in the $\Displays D$ table irrespective of its size.
	Note: The <code>save_copy_max_size</code> parameter clause is effective only when the <code>save_copy</code> parameter clause is specified.



Table 2-39 (Cont.) BASIC_STORAGE Attributes

Attribute	Attribute Value
separate_offsets	Parameter clause to improve the query performance for the CONTEXT index that is extensively used for IO operations, and whose queries are mainly single-word or boolean queries. Creating an index with the SEPARATE_OFFSETS index option requires the CREATE TRIGGER privilege, as a temporary trigger is created during the indexing process.
	Set it to T to enable the SEPARATE_OFFSETS index option for the CONTEXT index. The default is F .
	Note : The SEPARATE_OFFSETS index option is not supported for local Oracle Text search index.
single_byte	Storage option for better performance if all the indexed data that is known in advance is single-byte.
	When set to TRUE, all the data is treated as a single-byte (8-bit) data and the character set is irrelevant during indexing and querying. Ensure that no character in the data set crosses the single-byte (8-bit) limit. The default is FALSE.
small_r_row	Storage attribute to reduce the size of \$R row. It improves DML and query performance during parallel DML and query workload. It reduces lock contention during DMLs, thus improving the DML performance.
sn_table_clause	Parameter clause for dr\$indexname\$SN table creation. Specify the storage and tablespace clauses to add at the end of the internal CREATE TABLE statement. The default clause is: 'LOB (VAL_INFO) STORE AS (CACHE)'.
sn_index_clause	Parameter clause for dr\$indexname\$SNI table creation. Specify the storage and tablespace clauses to add at the end of the internal CREATE INDEX statement.
sd_table_clause	Parameter clause for dr\$indexname\$SD table creation. Specify the storage and tablespace clauses to add at the end of the internal CREATE TABLE statement. The default clause is: 'LOB (VAL_INFO) STORE AS (CACHE)'.
sd_index_clause	Parameter clause for dr\$indexname\$SDI table creation. Specify the storage and tablespace clauses to add at the end of the internal CREATE INDEX statement.
sv_table_clause	Parameter clause for dr\$indexname\$SV table creation. Specify the storage and tablespace clauses to add at the end of the internal CREATE TABLE statement. The default clause is: 'LOB (VAL_INFO) STORE AS (CACHE)'.
sv_index_clause	Parameter clause for dr\$indexname\$SVI table creation. Specify the storage and tablespace clauses to add at the end of the internal CREATE INDEX statement.
sr_table_clause	Parameter clause for dr\$indexname\$SR table creation. Specify the storage and tablespace clauses to add at the end of the internal CREATE TABLE statement. The default clause is: 'LOB (VAL_INFO) STORE AS (CACHE)'.
sr_index_clause	Parameter clause for dr\$indexname\$SRI table creation. Specify the storage and tablespace clauses to add at the end of the internal CREATE INDEX statement.
sbd_table_clause	Parameter clause for dr\$indexname\$SBD table creation. Specify the storage and tablespace clauses to add at the end of the internal CREATE TABLE statement. The default clause is: 'LOB (VAL_INFO) STORE AS (CACHE)'.
sbd_index_clause	Parameter clause for dr\$indexname\$SBDI table creation. Specify the storage and tablespace clauses to add at the end of the internal CREATE INDEX statement.
sbf_table_clause	Parameter clause for dr\$indexname\$SBF table creation. Specify the storage and tablespace clauses to add at the end of the internal CREATE TABLE statement. The default clause is: `LOB(VAL_INFO) STORE AS (CACHE)'.



Table 2-39 (Cont.) BASIC_STORAGE Attributes

Attribute	Attribute Value
sbf_index_clause	Parameter clause for dr\$indexname\$SBFI table creation. Specify the storage and tablespace clauses to add at the end of the internal CREATE INDEX statement.
st_table_clause	Parameter clause for dr\$indexname\$ST table creation. Specify the storage and tablespace clauses to add at the end of the internal CREATE TABLE statement. The default clause is: 'LOB (VAL_INFO) STORE AS (CACHE)'.
st_index_clause	Parameter clause for dr\$indexname\$STI table creation. Specify the storage and tablespace clauses to add at the end of the internal CREATE INDEX statement.
stz_table_clause	Parameter clause for dr\$indexname\$STZ table creation. Specify the storage and tablespace clauses to add at the end of the internal CREATE TABLE statement. The default clause is: 'LOB (VAL_INFO) STORE AS (CACHE)'.
stz_index_clause	Parameter clause for dr\$indexname\$STZI table creation. Specify the storage and tablespace clauses to add at the end of the internal CREATE INDEX statement.
sids_table_clause	Parameter clause for dr\$indexname\$SIDS table creation. Specify the storage and tablespace clauses to add at the end of the internal CREATE TABLE statement. The default clause is: 'LOB (VAL_INFO) STORE AS (CACHE)'.
sids_index_clause	Parameter clause for dr\$indexname\$SIDSI table creation. Specify the storage and tablespace clauses to add at the end of the internal CREATE INDEX statement.
siym_table_clause	Parameter clause for dr\$indexname\$SIYM table creation. Specify the storage and tablespace clauses to add at the end of the internal CREATE TABLE statement. The default clause is: 'LOB (VAL_INFO) STORE AS (CACHE)'.
siym_index_clause	Parameter clause for dr\$indexname\$SIYMI table creation. Specify the storage and tablespace clauses to add at the end of the internal CREATE INDEX statement.
stage_itab	Switch to improve the query performance for the ${\tt CONTEXT}$ index that is extensively used for DML operations.
	When the STAGE_ITAB index option is disabled, then when a new document is added to the index, SYNC_INDEX is called to make the documents searchable. This creates new rows in the \$I table, thus increasing the fragmentation in the \$I table. This leads to the deterioration of the query performance.
	When the STAGE_ITAB index option is enabled, the information about the new documents is stored in the \$G staging table, and not in the \$I table. This ensures that the \$I table does not get fragmented, and thus does not deteriorate the query performance.
	When the STAGE_ITAB index option is enabled, the \$H btree index is also created on the \$G table. The \$G table and \$H btree index are equivalent to the \$I table and \$X btree index.
	Set $stage_itab$ to YES to enable the $STAGE_ITAB$ index option for the CONTEXT index. The default is NO.
stage_itab_auto_opt	New storage option to enable automatic background optimize merge. stage_itab and stage_itab_auto_opt must be set to TRUE to enable automatic background optimize merge.
	Setting stage_itab_auto_opt to TRUE is not supported when stage_itab_max_rows is set to 0 as the zero value disables row movement from the \$G table to the \$I table.

Table 2-39 (Cont.) BASIC_STORAGE Attributes

Attribute	Attribute Value			
stage_itab_max_rows	Storage option to ensure that the \$G (stage_itab) table fits into the KEEP pool and also that the \$G table does not get filled up too frequently. This option is also required to ensure that \$G does not grow too big and start slowing down the query and the index synchronization performance.			
	When the number of rows in the \$G table exceeds this setting, a process is started to move all data from the \$G table to the \$I table, optimizing the data as it is moved. Note that this may cause certain SYNC operations or commits if SYNC(ON COMMIT) is used to take an unexpectedly long time because they may be moving many \$G rows which have been inserted by other processes. If this is unacceptable, set stage_itab_max_rows to 0 and use an auto optimization job instead.			
	When scheduling an auto optimization job, set <code>stage_itab_max_rows</code> to 0 to disable the automatic merging that now happens through sync index.			
	If stage_itab_max_rows is not set to 0 and an attempt is made to schedule an auto optimization job, then an error occurs.			
	You can set <pre>stage_itab_max_rows</pre> to either 0 or any value greater than or equal to 1000. The default value is 10K. A system with a very heavy DML load (inserts, deletes, and updates) but a low query load might benefit from a larger value as this reduces the number of merge operations which are necessary. For such indexes, Oracle recommends a value of 100K to 1 million.			
	If you set the value to 0 the automatic background merge is turned off. In this case, you must manually run $CTX_DDL.OPTIMIZE_INDEX$ in MERGE mode to move rows from the \$G staging table to the \$I permanent index table.			
	With stage_itab, when queries are run during heavy DML operations, Oracle Database can issue the following error: ORA-08176 consistent read failure; rollback data not available. In such cases, increase the size of the UNDO tablespace and the UNDO_RETENTION initialization parameter.			
stage_itab_parallel	New storage option controls the degree of parallelism used to merge rows from the stage_itab (\$G table) back to the \$I table when the stage_itab_max_rows limit is hit.			
	The default value is 16 for the degree of parallelism.			
u_table_clause	Specify the storage and tablespace clauses to add at the end of the internal CREATE TABLE statement. The \$U table keeps track of concurrent updates.			

Related Topics

SYNC_INDEX
Synchronizes the index to process inserts, updates, and deletes to the base table.

2.7.1.2 BASIC_STORAGE Default Behavior

By default, <code>BASIC_STORAGE</code> attributes are not set. In such cases, the Text index tables are created in the index owner's default tablespace. Consider the following statement, entered by user <code>IUSER</code>, with no <code>BASIC_STORAGE</code> attributes set:

create index IOWNER.idx on TOWNER.tab(b) indextype is ctxsys.context;

In this example, the text index is created in IOWNER's default tablespace.

2.7.1.3 BASIC STORAGE Examples

The following examples specify that the index tables are to be created in the foo tablespace with an initial extent of 1K:

```
begin
ctx ddl.create preference('mystore', 'BASIC STORAGE');
ctx ddl.set attribute('mystore', 'I TABLE CLAUSE',
                        'tablespace foo storage (initial 1K)');
ctx ddl.set attribute('mystore', 'K TABLE CLAUSE',
                        'tablespace foo storage (initial 1K)');
ctx ddl.set attribute('mystore', 'R TABLE CLAUSE',
                        'tablespace users storage (initial 1K) lob
                         (data) store as (disable storage in row cache)');
ctx ddl.set attribute('mystore', 'N TABLE CLAUSE',
                        'tablespace foo storage (initial 1K)');
ctx ddl.set attribute('mystore', 'I INDEX CLAUSE',
                        'tablespace foo storage (initial 1K) compress 2');
ctx ddl.set attribute('mystore', 'P TABLE CLAUSE',
                        'tablespace foo storage (initial 1K)');
ctx ddl.set attribute('mystore', 'S TABLE CLAUSE',
                        'tablespace foo storage (initial 1K)');
ctx ddl.set attribute('mystore', 'U TABLE CLAUSE',
                        'tablespace foo storage (initial 1K)');end;
```

The following example adds to the end of the internal table that is created.

```
exec ctx_ddl.create_preference('sto', 'basic_storage');
exec ctx_ddl.set_attribute('sto', 'e_table_clause', 'tablespace foo');
```

The following example uses query_filter_cache_size storage parameter for a partitioned index:

```
exec ctx_ddl.create_preference('fcs', 'basic_storage');
exec ctx_ddl.set_attribute('fcs', 'query_filter_cache_size', '100000000');

create table fc(id number primary key, txt varchar2(255))
partition by range (id)
(
         partition p1 values less than (25),
         partition p2 values less than (50),
         partition p3 values less than (75)
);

create index fci on fc(txt) indextype is ctxsys.context
    local (
    partition p1,
    partition p2,
    partition p3) parameters('storage fcs memory 49M sync (on commit)');
```

The query filter cache is an index level storage preference. The storage preference for the query filter cache can be set at partition level only if this is also set at the index level.

```
select count(*) from fc partition (p1) where contains(txt,'ctxfiltercache((hello))')>0;
```

```
Note:
```

Starting in Oracle Database Release 21c, CTXFILTERCACHE is deprecated, and also CTX_FILTER_CACHE_STATISTICS and QUERY_FILTER_CACHE_SIZE.

SINGLE_BYTE Data Indexing Storage Attribute

Syntax

```
ctx ddl.set attribute(storage pref name, 'SINGLE BYTE', BOOLEAN);
```

storage_pref_name

Specify the first argument as the storage preference name.

SINGLE BYTE

Specify the storage attribute name as SINGLE BYTE or single byte.

BOOLEAN

Indicate whether the attribute is set. By default, the value is FALSE. It implies that the database character set identifies whether the documents are stored as single-byte or multi-byte.

The following example sets the storage preference and enables the <code>single_byte</code> storage attribute:

```
exec ctx_ddl.create_preference('mysto', 'basic_storage');
ctx_ddl.set_attribute('mysto', 'single_byte', 'TRUE');
```

SMALL_R_ROW Storage Attribute

Syntax

```
ctx ddl.set attribute(storage pref name, 'SMALL R ROW', BOOLEAN);
```

storage_pref_name

Specify the first argument as the storage preference name.

SMALL R ROW

Specify the storage attribute name as SMALL R ROW or small r row..

BOOLEAN

Indicate whether the attribute is set. By default, the value is TRUE.

The following example sets the storage preference and enables the small_r_row storage attribute:

```
begin
ctx_ddl.create_preference('sto', 'basic_storage');
ctx_ddl.set_attribute('sto', 'small_r_row', 'T',
end;
```

To enable or disablesmall r row feature on an existing index:

```
ALTER INDEX index name rebuild PARAMETERS ('replace storage sto');
```

By default, small r row=TRUE, however, for earlier releases, small r row=FALSE.

2.8 Section Group Types

To enter WITHIN queries on document sections, you must create a section group before you define your sections. Specify your section group in the parameter clause of CREATE INDEX.

This section contains the following topics.

- Section Group Types for Creating a Section Group
- Section Group Examples for HTML, XML, and JSON Enabled Documents

2.8.1 Section Group Types for Creating a Section Group

To create a section group, you can specify one of the following group types with the CTX_DDL.CREATE_SECTION_GROUP procedure.

Table 2-40 Section Group Types

Туре	Description	
NULL_SECTION_GROUP	Use this group type when you define no sections or when you define only SENTENCE or PARAGRAPH sections. This is the default.	
BASIC_SECTION_GROUP	Use this group type for defining sections where the start and end tags are of the form <a> and .	
	Note: This group type does not support input such as unbalanced parentheses, comments tags, and attributes. Use HTML_SECTION_GROUP for this type of input.	
HTML_SECTION_GROUP	Use this group type for indexing HTML documents and for defining sections in HTML documents.	
JSON_SECTION_GROUP	Use this group to create a JSON enabled context index. The <code>JSON</code> <code>ENABLE</code> attribute cannot be used with <code>XML</code> <code>ENABLE</code> . A section group can only be marked as <code>JSON</code> <code>ENABLE</code> . If it is already marked with <code>XML</code> <code>ENABLE</code> , then the path section group cannot be used for <code>JSON</code> <code>ENABLE</code> and vice versa.	
XML_SECTION_GROUP	Use this group type for indexing XML documents and for defining sections in XML documents. All sections to be indexed must be manually defined for this group.	



Table 2-40 (Cont.) Section Group Types

Туре	Description
AUTO_SECTION_GROUP	Use this group type to automatically create a zone section for each start-tag/end-tag pair in an XML document. The section names derived from XML tags are case sensitive as in XML.
	Attribute sections are created automatically for XML tags that have attributes. Attribute sections are named in the form tag@attribute.
	Special sections can be added to AUTO_SECTION_GROUP for WITHIN SENTENCE and WITHIN PARAGRAPH searches. Once a sentence or paragraph section is added to the AUTO_SECTION_GROUP, sections with corresponding tag names 'sentence' or 'paragraph' (case insensitive) are treated as stop sections.
	Stop sections, empty tags, processing instructions, and comments are not indexed.
	The following limitations apply to automatic section groups:
	 You cannot add zone, field, sdata, or special sections to an automatic section group.
	 You can define a stop section that applies only to one particular type; that is, if you have two different XML DTDs, both of which use a tag called FOO, you can define (TYPE1) FOO to be stopped, but (TYPE2) FOO to not be stopped.
	 The length of the indexed tags, including prefix and namespace, cannot exceed 64 bytes. Tags longer than this are not indexed.
PATH_SECTION_GROUP	Use this group type to index XML documents. Behaves like the AUTO_SECTION_GROUP.
	The difference is that with this section group you can do path searching with the INPATH and HASPATH operators. Queries are also casesensitive for tag and attribute names. Stop sections are not allowed.
NEWS_SECTION_GROUP	Use this group for defining sections in newsgroup formatted documents according to RFC 1036.



Starting with Oracle Database 18c, use of <code>NEWS_SECTION_GROUP</code> is deprecated in Oracle Text. Use external processing instead.

If you want to index USENET posts, then preprocess the posts to use ${\tt BASIC_SECTION_GROUP} \ or \ {\tt HTML_SECTION_GROUP} \ within \ Oracle \ Text. \ USENET \ is \ rarely \ used \ commercially.$

2.8.2 Section Group Examples for HTML, XML, and JSON Enabled Documents

The examples show the use of section groups in HTML and XML documents, and in JSON enabled documents. See Table 2-40 for a summary.

This section contains the following examples:

- Creating Section Groups in HTML Documents
- Creating Sections Groups in XML Documents



- Automatic Sectioning in XML Documents
- Creating JSON Section Groups for JSON Search Index
- Using JSON Search Index with JSON_TEXTCONTAINS
- Using JSON Search Index with JSON EXISTS

2.8.2.1 Creating Section Groups in HTML Documents

The following statement creates a section group called htmgroup with the HTML group type.

```
begin
ctx_ddl.create_section_group('htmgroup', 'HTML_SECTION_GROUP');
end;
```

You can optionally add sections to this group using the procedures in the CTX_DDL package, such as CTX_DDL.ADD_SPECIAL_SECTION or CTX_DDL.ADD_ZONE_SECTION. To index your documents, enter a statement such as:

create index myindex on docs(htmlfile) indextype is ctxsys.context
parameters('filter ctxsys.null filter section group htmgroup');



For more information on section groups, see CTX DDL Package

2.8.2.2 Creating Sections Groups in XML Documents

The following statement creates a section group called xmlgroup with the XML_SECTION_GROUP group type.

```
begin
ctx_ddl.create_section_group('xmlgroup', 'XML_SECTION_GROUP');
end:
```

You can optionally add sections to this group using the procedures in the CTX_DDL package, such as CTX_DDL.ADD_ATTR_SECTION or CTX_DDL.ADD_STOP_SECTION. To index your documents, enter a statement such as:

```
create index myindex on docs(htmlfile) indextype is ctxsys.context
parameters('filter ctxsys.null filter section group xmlgroup');
```

See Also:

For more information on section groups, see CTX_DDL Package

2.8.2.3 Automatic Sectioning in XML Documents

The following statement creates a section group called auto with the AUTO_SECTION_GROUP group type. This section group automatically creates sections from tags in XML documents.

begin



```
ctx_ddl.create_section_group('auto', 'AUTO_SECTION_GROUP');
end;

CREATE INDEX myindex on docs(htmlfile) INDEXTYPE IS ctxsys.context
PARAMETERS('filter ctxsys.null filter section group auto');
```

2.8.2.4 Creating JSON Section Groups for JSON Search Index

The following example creates a JSON enabled text index.

```
create index json_ctx_idx on customers (customer
_info)
indextype is ctxsys.context
parameters ('section group CTXSYS.JSON_SECTION_GROUP');
```

2.8.2.5 Using JSON Search Index with JSON_TEXTCONTAINS

The following example searches for customers having keyword "gold" in the description.

```
select customer_info
from customers
where JSON_TEXTCONTAINS(customer_info, '$.description', 'gold');
```

2.8.2.6 Using JSON Search Index with JSON_EXISTS

Find JSON enabled data.

```
select customer_info from customers
where JSON EXISTS(customer info, '$.dataplan');
```

2.9 Classifier Types

The following classifier types are used to create preferences for CTS_CLS.TRAIN and CTXRULE index creation:

- RULE_CLASSIFIER
- SVM CLASSIFIER
- SENTIMENT CLASSIFIER



In Oracle Database Express Edition (Oracle Database XE), $RULE_CLASSIFIER$, $SVM_CLASSIFIER$, and $SENTIMENT_CLASSIFIER$ are not supported because the Data Mining option is not available. This is also true for $KMEAN_CLUSTERING$.

2.9.1 RULE_CLASSIFIER

Use the RULE_CLASSIFIER type for creating preferences for the query rule generating procedure, CTX_CLS.TRAIN and for CTXRULE creation. The rules generated with this type are essentially query strings and can be easily examined. The queries generated by this classifier

can use the AND, NOT, or ABOUT operators. The WITHIN operator is supported for queries on field sections only.

Table 2-41 lists the attributes for the RULE CLASSIFIER type.

Table 2-41 RULE_CLASSIFIER Attributes

Attribute	Data Type	Default	Min Value	Max Value	Description
THRESHOLD	I	50	1	99	Specify threshold (in percentage) for rule generation. One rule is output only when its confidence level is larger than threshold.
MAX_TERMS	I	100	20	2000	For each class, a list of relevant terms is selected to form rules. Specify the maximum number of terms that can be selected for each class.
MEMORY_SIZE	I	500	10	4000	Specify memory usage for training in MB. Larger values improve performance.
NT_THRESHOLD	F	0.001	0	0.90	Specify a threshold for term selection. There are two thresholds guiding two steps in selecting relevant terms. This threshold controls the behavior of the first step. At this step, terms are selected as candidate terms for the further consideration in the second step. The term is chosen when the ratio of the occurrence frequency over the number of documents in the training set is larger than this threshold.
TERM_THRESHOLD	I	10	0	100	Specify a threshold as a percentage for term selection. This threshold controls the second step term selection. Each candidate term has a numerical quantity calculated to imply its correlation with a given class. The candidate term will be selected for this class only when the ratio of its quantity value over the maximum value for all candidate terms in the class is larger than this threshold.
PRUNE_LEVEL	I	75	0	100	Specify how much to prune a built decision tree for better coverage. Higher values mean more aggressive pruning and the generated rules will have larger coverage but less accuracy.

2.9.2 SVM_CLASSIFIER

Use the SVM_CLASSIFIER type for creating preferences for the rule generating procedure, CTX_CLS.TRAIN, and for CTXRULE creation. This classifier type represents the Support Vector Machine method of classification and generates rules in binary format. Use this classifier type when you need high classification accuracy.

This type has the following attributes:



Table 2-42 SVM_CLASSIFIER Attributes

Attribute Name	Data Type	Default	Min Value	Max Value	Description
MAX_DOCTERMS	I	50	10	8192	Specify the maximum number of terms representing one document.
MAX_FEATURES	1	3,000	1	100,000	Specify the maximum number of distinct features.
THEME_ON	В	FALSE	NULL	NULL	Specify TRUE to use themes as features.
TOKEN_ON	В	TRUE	NULL	NULL	Specify TRUE to use regular tokens as features.
STEM_ON	В	FALSE	NULL	NULL	Specify TRUE to use stemmed tokens as features. This only works when turning INDEX_STEM on for the lexer.
MEMORY_SIZE	1	500	10	4000	Specify approximate memory size in MB.
SECTION_WEIGHT	1	2	0	100	Specify the occurrence multiplier for adding a term in a field section as a normal term. For example, by default, the term cat in " <a>cat " is a field section term and is treated as a normal term with occurrence equal to 2, but you can specify that it be treated as a normal term with a weight up to 100. SECTION_WEIGHT is only meaningful when the index policy specifies a field section.

2.9.3 SENTIMENT_CLASSIFIER

Use the <code>SENTIMENT_CLASSIFIER</code> type to create a preference for sentiment analysis queries. This classifier specifies preferences associated with a user-defined sentiment classifier preference. You must define a preference of this type before you use the $CTX_CLS.SA_TRAIN_MODEL$ procedure to train the user-defined sentiment classifier.

Table 2-43 lists the attributes for the SENTIMENT_CLASSIFIER type.

Table 2-43 SENTIMENT_CLASSIFIER Attributes

Attribute	Data Type	Default	Minimum Value	Maximum Value	Description
MAX_DOCTERMS	I	50	10	8192	Specify the maximum number of distinct terms representing one document
MAX_FEATURES	1	3000	1	100000	Specify the maximum number of distinct features used to build a sentiment classifier
THEME_ON	В	False			Specify if themes must be extracted as features



Table 2-43 (Cont.) SENTIMENT_CLASSIFIER Attributes

Attribute	Data Type	Default	Minimum Value	Maximum Value	Description
TOKEN_ON	В	True			Specify if tokens must be extracted as features
STEM_ON	В	True			Specify if stemmed tokens must be extracted as features
MEMORY_SIZE	1	500	10	4000	Specify the typical memory size, in MB, used to build the sentiment classifier.
SECTION_WEIGHT	1	2	0	100	Specify the integer multiplier for term occurrence within a field section
NUM_ITERATIONS	I	600			Specify the maximum number of iterations for which the sentiment classifier is run before it converges

See Also:

Oracle Text Application Developer's Guide for an example of using the ${\tt SENTIMENT}$ CLASSIFIER type

2.10 Cluster Types

This section describes the cluster types used for creating preferences for the ${\tt CTX}$ CLS.CLUSTERING procedure.

KMEAN_CLUSTERING

Note:

In Oracle Database Express Edition (Oracle Database XE), $\texttt{KMEAN_CLUSTERING}$ is not supported because the Data Mining option is not available. This is also true for $\texttt{RULE_CLASSIFIER}$ and $\texttt{SVM_CLASSIFIER}$.

See Also:

For more information about clustering, see "CLUSTERING" in CTX_CLS Package as well as the *Oracle Text Application Developer's Guide*

2.10.1 KMEAN_CLUSTERING

The KMEAN CLUSTERING clustering type has the attributes listed in Table 2-44.

Table 2-44 KMEAN_CLUSTERING Attributes

Attribute Name	Data Type	Default	Min Value	Max Value	Description
MAX_DOCTERMS	I	50	10	8192	Specify the maximum number of distinct terms representing one document.
MAX_FEATURES	1	3,000	1	500,000	Specify the maximum number of distinct features.
THEME_ON	В	FALSE	NULL	NULL	Specify TRUE to use themes as features.
TOKEN_ON	В	TRUE	NULL	NULL	Specify TRUE to use regular tokens as features.
STEM_ON	В	FALSE	NULL	NULL	Specify TRUE to use stemmed tokens as features. This only works when turning INDEX_STEM on for the lexer.
MEMORY_SIZE	1	500	10	4000	Specify approximate memory size in MB.
SECTION_WEIGHT	1	2	0	100	Specify the occurrence multiplier for adding a term in a field section as a normal term. For example, by default, the term <i>cat</i> in " <a>cat " is a field section term and is treated as a normal term with occurrence equal to 2, but you can specify that it be treated as a normal term with a weight up to 100. SECTION_WEIGHT is only meaningful when the index policy specifies a field section.
CLUSTER_NUM	1	200	2	20000	Specify the total number of leaf clusters to be generated.

2.11 Stoplists

Stoplists identify the words in your language that are not to be indexed. In English, you can also identify stopthemes that are not to be indexed.

- Multi-Language Stoplists
- Creating Stoplists
- Supplied Stoplists
- Modifying the Default Stoplist

2.11.1 Multi-Language Stoplists

You can create multi-language stoplists to hold language-specific stopwords. A multi-language stoplist is useful when you use the MULTI_LEXER to index a table that contains documents in different languages, such as English, German.



To create a multi-language stoplist, use the CTX_DLL.CREATE_STOPLIST procedure and specify a stoplist type of MULTI_STOPLIST. Add language specific stopwords with CTX DDL.ADD_STOPWORD.

At indexing time, the language column of each document is examined, and only the stopwords for that language are eliminated. At query time, the session language setting determines the active stopwords, like it determines the active lexer when using the multi-lexer.

2.11.2 Creating Stoplists

Create your own stoplists using the CTX_DLL.CREATE_STOPLIST procedure. With this procedure you can create a BASIC_STOPLIST for single language stoplist, or you can create a MULTI STOPLIST for a multi-language stoplist.

When you create your own stoplist, you must specify it in the parameter clause of CREATE INDEX.

To create stoplists for Chinese or Japanese languages, use the CHINESE_LEXER or JAPANESE_LEXER respectively, and update the appropriate lexicon to be @contained_such_stopwords.

2.11.3 Supplied Stoplists

By default, the system indexes text using the Oracle Text supplied stoplists that correspond to your database language.

A stoplist is a list of stopwords that do not get indexed. These are usually common words in a language, such as *this*, *that*, and *can* in English. By default, all such words are defined in the Oracle Text supplied stoplists. You can customize these stoplists or update the stopwords based on your requirements.

Supported Languages and Stoplists Location

The Oracle Text supplied stoplists contain a list of stopwords, which are provided as defaults for all BASIC_LEXER and AUTO_LEXER supported languages. These stopwords are automatically loaded during installation or upgrade for the chosen database language.

The default stoplists (along with other default preferences) are defined in the administration (SQL) files, which are located in the $\$ORACLE_HOME/ctx/admin$ directory. These SQL files are named drdefLANG.sql, where LANG specifies the language code. For example, the default stoplist for French (language code: f) is defined in the $\$ORACLE_HOME/ctx/admin/drdeff.sql$ file.

The source files for these default stoplists contain a list of stopwords, and are located in the <code>\$ORACLE_HOME/ctx/data/stoplist</code> directory. These source files are named <code>drstoplang.txt</code>, where <code>Lang</code> specifies the language code. The contents of the source files are the extracted terms from the <code>drdefLang.sql</code> files.

For a list of all languages (and their language codes) in which default stoplists are supplied, see Multilingual Features Matrix.

How to Load Your Own Stoplists

By default, only one $\mathtt{drdef}\mathit{LANG}.sql$ file is loaded during installation or upgrade based on the database language that you choose. You can call the $\mathtt{CTX_DDL}.LOAD_STOPLIST$ procedure to customize your stoplist or modify the default list of stopwords.



Unlike CTX_DDL.ADD_STOPWORD (which adds a single stopword per call), CTX_DDL.LOAD_STOPLIST takes a source file of stopwords for your specified language (from \$ORACLE HOME/ctx/data/stoplist/drstopLANG.txt) and loads to your stoplist.

Related Topics

LOAD STOPLIST

Use this procedure to load a source file of stopwords to your stoplist for the required language.

Multilingual Features Matrix
 These are the multilingual features for all supported languages.

2.11.4 Modifying the Default Stoplist

The default stoplist is always named CTXSYS.DEFAULT_STOPLIST. Use this procedure to modify this stoplist.

- CTX_DDL.ADD_STOPWORD
- CTX_DDL.ADD_STOPTHEME
- CTX_DDL.ADD_STOPCLASS
- CTX_DDL.LOAD_STOPLIST
- CTX_DDL.REMOVE_STOPWORD

When you modify CTXSYS.DEFAULT_STOPLIST with the CTX_DDL package, you must re-create your index for the changes to take effect.

Dynamic Addition of Stopwords

You can *add* stopwords dynamically to a default or custom stoplist with ALTER INDEX . When you add a stopword dynamically, you need not re-index, because the word immediately becomes a stopword and is removed from the index.



Even though you can dynamically add stopwords to an index, you cannot dynamically remove stopwords. To remove a stopword, you must use CTX DDL.REMOVE STOPWORD, drop your index and re-create it.

Related Topics

ALTER INDEX

Use the ALTER INDEX statement to change or rebuild an existing index, such as Oracle Text index, Oracle Text search index, JSON search index, XML search index, or hybrid vector index.

2.12 System-Defined Preferences

When you install Oracle Text, some indexing preferences are created. You can use these preferences in the parameter clause of CREATE INDEX or define your own.

The default index parameters are mapped to some of the system-defined preferences described in this section.

See Also:

For more information about default index parameters, see "Default Index Parameters"

System-defined preferences are divided into the following categories:

- Data Storage Preferences
- Filter Preferences
- Lexer Preferences
- Section Group Preferences
- Stoplist Preferences
- Storage Preferences
- Wordlist Preferences

2.12.1 Data Storage Preferences

This section discusses the types associated with data storage preferences.

- The CTXSYS.DEFAULT_DATASTORE preference uses the DIRECT_DATASTORE type. Use
 this preference to create indexes for text columns in which the text is stored directly in the
 column.
- The CTXSYS.FILE DATASTORE preference uses the FILE_DATASTORE type.
- The CTXSYS.URL DATASTORE preference uses the URL_DATASTORE type.

2.12.2 Filter Preferences

This section discusses the types associated with filtering preferences.

- The CTXSYS.NULL FILTER preference uses the NULL_FILTER type.
- The CTXSYS.AUTO FILTER preference uses the AUTO_FILTER type.

2.12.3 Lexer Preferences

This section discusses the types associated with *lexer* preferences.

- CTXSYS.DEFAULT_LEXER
- CTXSYS.DEFAULT_EXTRACT_LEXER
- CTXSYS.BASIC_LEXER

2.12.3.1 CTXSYS.DEFAULT_LEXER

The CTXSYS.DEFAULT_LEXER default lexer depends on the language used at install time.

The following sections describe the default settings for CTXSYS.DEFAULT_LEXER for each language.

American and English Language Settings

If your language is English, this preference uses the BASIC_LEXER with the index themes attribute disabled.

Danish Language Settings

If your language is Danish, this preference uses the BASIC_LEXER with the following option enabled:

- Alternate spelling (alternate_spelling attribute set to DANISH)
- Dutch Language Settings

If your language is Dutch, this preference uses the BASIC_LEXER with the following options enabled:

- composite indexing (composite attribute set to DUTCH)
- German and German DIN Language Settings

If your language is German, then this preference uses the BASIC_LEXER with the following options enabled:

- Case-sensitive indexing (mixed case attribute enabled)
- Composite indexing (composite attribute set to GERMAN)
- Alternate spelling (alternate spelling attribute set to GERMAN)
- Bokmal (Norwegian), Finnish, Nynorsk (Norwegian), and Swedish Language Settings
 If your language is Bokmal (Norwegian), Finnish, Nynorsk (Norwegian), or Swedish, this preference uses the BASIC LEXER with the following option enabled:
 - Alternate spelling (alternate_spelling attribute set to SWEDISH)
- Japanese Language Settings

If your language is Japanese, this preference uses the JAPANESE VGRAM LEXER.

Korean Language Settings

If your language is Korean, this preference uses the KOREAN_MORPH_LEXER . All attributes for the KOREAN MORPH LEXER are enabled.

Chinese Language Settings

If your language is Simplified or Traditional Chinese, this preference uses the CHINESE_VGRAM_LEXER.

Other Languages

For all other languages not listed in this section, this preference uses the BASIC_LEXER with no attributes set.



To learn more about these options, see "BASIC LEXER"

2.12.3.2 CTXSYS.DEFAULT_EXTRACT_LEXER

The CTXSYS.DEFAULT_EXTRACT_LEXER preference uses AUTO_LEXER and includes all Oracle-supplied features (rules, dictionary, etc.). CTXSYS.DEFAULT_EXTRACT_LEXER uses AUTO_LEXER with the following options:

- alternate spelling is NONE
- base letter is NO
- mixed_case is YES
- <> printjoin is '-*' <>

2.12.3.3 CTXSYS.BASIC LEXER

The CTXSYS.BASIC LEXER preference uses the BASIC_LEXER.

2.12.4 Section Group Preferences

This section discusses the types associated with section group preferences.

- The CTXSYS.NULL SECTION GROUP preference uses the NULL SECTION GROUP type.
- The CTXSYS.HTML SECTION GROUP preference uses the HTML SECTION GROUP type.
- The CTXSYS.JSON_SECTION_GROUP preference uses the PATH_SECTION_GROUP type.
- The CTXSYS.AUTO SECTION GROUP preference uses the AUTO SECTION GROUP type.
- The CTXSYS.PATH SECTION GROUP preference uses the PATH SECTION GROUP type.

Here is the list of default section groups that are created:

- The CTXSYS.XQUERY_SEC_GROUP preference evaluates not only xquery full text expressions but also the xquery range expressions.
- The CTXSYS.XQFT SEC GROUP preference evaluates only xquery full text expressions.

2.12.5 Stoplist Preferences

This section discusses the types associated with stoplist preferences.

- The CTXSYS.DEFAULT_STOPLIST stoplist preference defaults to the stoplist of your database language.
- The CTXSYS.EMPTY STOPLIST stoplist has no words.



For a complete list of the stop words in the supplied stoplists, see Supplied Stoplists.

2.12.6 Storage Preferences

This section discusses the types associated with storage preferences.

The CTXSYS.DEFAULT STORAGE storage preference uses the BASIC_STORAGE type.

Here are the storage preferences:

- The CTXSYS.XQFT_LOW preference disables the persistence of secondary XML representation into \$D table to save index storage space.
 - xml save copy = FALSE



- xml forward enable = FALSE
- The CTXSYS.XQFT_MEDIUM preference enables the persistence of secondary XML representation into \$D table to reduce the time spent on post index xquery evaluation, if needed.
 - xml_save_copy = TRUExml forward enable = FALSE
- The CTXSYS.XQFT_HIGH preference enables the persistence of secondary XML
 representation into \$D table and forwards the index into \$D to reduce the time spent on post
 index xquery and xquery full text expression evaluation, if needed.
 - xml_save_copy = TRUE
 xml_forward_enable = TRUE

2.12.7 Wordlist Preferences

This section discusses the types associated with wordlist preferences.

The CTXSYS.DEFAULT_WORDLIST preference uses the language stemmer for your database language. If your language is not listed in Table 2-37, then this preference defaults to the NULL stemmer and the GENERIC fuzzy matching attribute.

2.13 System Parameters

This section describes the Oracle Text system parameters, which are divided into the following categories:

- General System Parameters
- Default Index Parameters
- Default Policy Parameters



2.13.1 General System Parameters

When you install Oracle Text, in addition to the system-defined preferences, the following system parameters are set:

Table 2-45 General System Parameters

System Parameter	Description
MAX_INDEX_MEMORY	This is the maximum indexing memory that can be specified in the parameter clause of CREATE INDEX and ALTER INDEX. The maximum value for this parameter is 256 GB.
DEFAULT_INDEX_MEMORY	This is the default indexing memory used with CREATE INDEX and ALTER INDEX. The default value for this parameter is 64 MB.



Table 2-45 (Cont.) General System Parameters

System Parameter	Description
LOG_DIRECTORY	This is the directory for CTX_OUTPUT log files.
CTX_DOC_KEY_TYPE	This is the default input key type, either ROWID or PRIMARY_KEY, for the CTX_DOC procedures. Set to ROWID at install time. See Also: CTX_DOC.SET_KEY_TYPE.

View system defaults by querying the CTX_PARAMETERS view. Change defaults using the CTX_ADM.SET_PARAMETER procedure.

2.13.2 Default Index Parameters

This section describes the index parameters that you can use when you create CONTEXT and CTXCAT indexes.

This section contains the following topics:

- CONTEXT Index Parameters
- CTXCAT Index Parameters
- CTXRULE Index Parameters

Viewing Default Values

View system defaults by querying the CTX_PARAMETERS view. For example, to see all parameters and values, enter the following statement:

SQL> SELECT par_name, par_value from ctx_parameters;

Changing Default Values

Change a default value using the CTX_ADM.SET_PARAMETER procedure to name another custom or system-defined preference to use as default.

2.13.2.1 CONTEXT Index Parameters

The following default parameters are used when you create a CONTEXT index and do not specify preferences in the parameter clause of CREATE INDEX. Each default parameter names a system-defined preference to use for data storage, filtering, lexing, and so on.

Table 2-46 Default CONTEXT Index Parameters

Parameter	Used When	Default Value
DEFAULT_DATASTORE	No datastore preference specified in parameter clause of CREATE INDEX.	CTXSYS.DEFAULT_DATASTORE



Table 2-46 (Cont.) Default CONTEXT Index Parameters

Parameter	Used When	Default Value
DEFAULT_FILTER_FILE	No filter preference specified in parameter clause of CREATE INDEX, and either of the following conditions is true:	CTXSYS.AUTO_FILTER
	 Your files are stored in external files (BFILES) or Specify a datastore preference that uses FILE_DATASTORE 	
DEFAULT_FILTER_BINARY	No filter preference specified in parameter clause of CREATE INDEX, and Oracle Text detects that the text column datatype is RAW, LONG RAW, or BLOB.	CTXSYS.AUTO_FILTER
DEFAULT_FILTER_TEXT	No filter preference specified in parameter clause of CREATE INDEX, and Oracle Text detects that the text column datatype is either LONG, VARCHAR2, VARCHAR, CHAR, or CLOB.	CTXSYS.NULL_FILTER
DEFAULT_SECTION_HTML	No section group specified in parameter clause of CREATE INDEX, and when either of the following conditions is true:	CTXSYS.HTML_SECTION_GROUP
	 Your datastore preference uses URL_DATASTORE or Your filter preference uses AUTO FILTER. 	
DEFAULT_SECTION_TEXT	No section group specified in parameter clause of CREATE INDEX, and when you do <i>not</i> use either URL_DATASTORE or AUTO_FILTER.	CTXSYS.NULL_SECTION_GROUP
DEFAULT_STORAGE	No storage preference specified in parameter clause of CREATE INDEX.	CTXSYS.DEFAULT_STORAGE
DEFAULT_LEXER	No lexer preference specified in parameter clause of CREATE INDEX.	CTXSYS.DERAULT_LEXER
DEFAULT_STOPLIST	No stoplist specified in parameter clause of CREATE INDEX.	CTXSYS.DEFAULT_STOPLIST
DEFAULT_WORDLIST	No wordlist preference specified in parameter clause of CREATE INDEX.	CTXSYS.DEFAULT_WORDLIST

See Also:

"System-Defined Preferences"



2.13.2.2 CTXCAT Index Parameters

These default parameters are used when you create a CTXCAT index with CREATE INDEX and do not specify any parameters in the parameter string.

The CTXCAT index supports only the index set, lexer, storage, stoplist, and wordlist parameters. Each default parameter names a system-defined preference.

Note:

The Oracle Text indextype CTXCAT is deprecated with Oracle Database 23ai. The indextype itself, and it's operator CTXCAT, can be removed in a future release. Both CTXCAT and the use of CTXCAT grammar as an alternative grammar for CONTEXT queries is deprecated. Instead, Oracle recommends that you use the CONTEXT indextype, which can provide all the same functionality, except that it is not transactional. Near-transactional behavior in CONTEXT can be achieved by using SYNC (ON COMMIT) or, preferably, SYNC (EVERY [time-period]) with a short time period.

CTXCAT was introduced when indexes were typically a few megabytes in size. Modern, large indexes, can be difficult to manage with CTXCAT. The addition of index sets to CTXCAT can be achieved more effectively by the use of FILTER BY and ORDER BY columns, or SDATA, or both, in the CONTEXT indextype. CTXCAT is therefore rarely an appropriate choice. Oracle recommends that you choose the more efficient CONTEXT indextype.

Table 2-47 Default CTXCAT Index Parameters

Parameter	Used When	Default Value
DEFAULT_CTXCAT_INDEX_SET	No index set specified in parameter clause of CREATE INDEX.	n/a
DEFAULT_CTXCAT_STORAGE	No storage preference specified in parameter clause of CREATE INDEX.	CTXSYS.DEFAULT_STORAGE
DEFAULT_CTXCAT_LEXER	No lexer preference specified in parameter clause of CREATE INDEX.	CTXSYS.DERAULT_LEXER
DEFAULT_CTXCAT_STOPLIST	No stoplist specified in parameter clause of CREATE INDEX.	CTXSYS.DEFAULT_STOPLIST
DEFAULT_CTXCAT_WORDLIST	No wordlist preference specified in parameter clause of CREATE INDEX.	CTXSYS.DEFAULT_WORDLIST
	Note that while you can specify a wordlist preference for CTXCAT indexes, most of the attributes do not apply, because the catsearch query language does not support wildcarding, fuzzy, and stemming. The only attribute that is useful is PREFIX_INDEX for Japanese data.	

Related Topics

System-Defined Preferences



2.13.2.3 CTXRULE Index Parameters

Table 2-48 lists the default parameters that are used when you create a CTXRULE index with CREATE INDEX and do not specify any parameters in the parameter string. The CTXRULE index supports only the lexer, storage, stoplist, and wordlist parameters. Each default parameter names a system-defined preference.

Table 2-48 Default CTXRULE Index Parameters

Parameter	Used When	Default Value
DEFAULT_CTXRULE_LEXER	No lexer preference specified in parameter clause of CREATE INDEX.	CTXSYS.DERAULT_LEXER
DEFAULT_CTXRULE_STORAGE	No storage preference specified in parameter clause of CREATE INDEX.	CTXSYS.DEFAULT_STORAGE
DEFAULT_CTXRULE_STOPLIST	No stoplist specified in parameter clause of CREATE INDEX.	CTXSYS.DEFAULT_STOPLIST
DEFAULT_CTXRULE_WORDLIST	No wordlist preference specified in parameter clause of CREATE INDEX.	CTXSYS.DEFAULT_WORDLIST
DEFAULT_CLASSIFIER	No classifier preference is specified in parameter clause.	RULE_CLASSIFIER



"System-Defined Preferences"

CTXRULE Index Limitations

The CTXRULE index does not support the following guery operators:

- Fuzzy
- Soundex

It also does not support the following BASIC WORDLIST attributes:

- SUBSTRING INDEX
- PREFIX INDEX

2.13.3 Default Policy Parameters

Policies in Oracle Text enable you to use document services without creating an index. For example, the document services might be filtering to generate a plain text or HTML version of a document, generating theme summaries or lists of themes, and highlighting.

Table 2-49 lists the default parameters when you create a policy and do not specify preferences when using CTX_DDL.CREATE_POLICY. Each default parameter names a system-defined preference to use for filtering, lexing, and so on.



Table 2-49 Default Policy Parameters for CTX_DDL.CREATE_POLICY

Parameter	Used When	Default Value
DEFAULT_FILTER_BINARY	No filter preference specified for CREATE_POLICY, and the document parameter of the document service is VARCHAR2 or CLOB datatype; BLOB or BFILE datatype.	CTXSYS.AUTO_FILTER
DEFAULT_FILTER_TEXT	No filter preference specified for CREATE_POLICY, and the document parameter of the document service is VARCHAR2 or CLOB datatype; BLOB or BFILE datatype.	CTXSYS.NULL_FILTER
DEFAULT_SECTION_HTML	No section group specified for CREATE_POLICY, and when your filter preference uses AUTO_FILTER.	CTXSYS.HTML_SECTION_GROUP
DEFAULT_SECTION_TEXT	No section_group specified for CREATE_POLICY, and when you do not use AUTO_FILTER.	CTXSYS.NULL_SECTION_GROUP
DEFAULT_LEXER	No lexer preference specified for CREATE_POLICY.	CTXSYS.DERAULT_LEXER
DEFAULT_STOPLIST	No stoplist specified for CREATE_POLICY.	CTXSYS.DEFAULT_STOPLIST
DEFAULT_WORDLIST	No wordlist preference specified for CREATE_POLICY.	CTXSYS.DEFAULT_WORDLIST

See Also:

- "System-Defined Preferences"
- "CREATE POLICY" for complete information

2.14 Token Limitations for Oracle Text Indexes

Starting with Oracle Database Release 18c, the indexed token maximum size is increased to 255 characters for single-byte character sets.

Before Oracle Database Release 18c, all Oracle Text index types except SDATA sections stored tokens in a table column of type VARCHAR2 (64 BYTE). Starting with Oracle Database Release 18c, all Oracle Text index types except CTXCAT and CTXRULE indexes store tokens in VARCHAR2 (255 BYTE) table column types. This change is an increase for the maximum size of an indexed token to 255 characters for single-byte character sets. The size increase is less with multibyte or variable-length character sets. Tokens longer than 255 bytes are truncated. Truncated tokens do not prevent searches on the whole token string. However, the system cannot distinguish between two tokens that have the same first 255 bytes.

Note:

Before Oracle Database Release 18c, tokens that were greater than 64 bytes were truncated to 64 bytes. After upgrading to Oracle Database Release 18c, the token tables are increased to 255 bytes from 64 bytes. Searches with more than 64 bytes in the search token (that is, any single word in search string) cannot find any tokens which were truncated to 64 bytes. To avoid this problem, rebuild the index. If you never use search tokens longer than 64 bytes, it is not necessary to rebuild the index.

SDATA sections store tokens in a table column of type VARCHAR2 (249 BYTE). CTXCAT and CTXRULE indexes store tokens in a table column of type VARCHAR2 (64 BYTE).

2.15 Auditing Oracle Text DR\$ Index Tables

You should consider creating audit policies for Oracle Text DR\$ index tables, especially if the base index table has sensitive information.

- About Auditing Oracle Text DR\$ Index Tables
- Configuring an Oracle Text DR\$ Index Tables Audit Policy
- Example: Auditing Update Actions on an Oracle Text DR\$ Index Table
- How Oracle Text DR\$ Index Table Entries Appear in the Audit Trail

2.15.1 About Auditing Oracle Text DR\$ Index Tables

You can audit actions on Oracle Text index tables (DR\$index), which can contain sensitive data.

The audit can capture actions that a user will perform on the index table. You should create a unified audit policy for the table that contains the sensitive data, as well as the Oracle Text index table for the column containing the sensitive data. Oracle Text index table names start with a prefix of DR\$.

Index tables that do not contain customer data do not need audit policies. Tables that you should consider creating audit policies for include the following:

- DR\$index name\$I (the main table that all users should protect)
- DR\$index name\$G (if present, stage_itab preference)
- DR\$index name\$P (if present, prefix index preference)
- DR\$index name\$0 (if present, forward index preference)
- DR\$index name\$D (if present, save copy preference)
- DR\$index name\$KG (if present, wildcard index preference)
- DR\$index_name\$SN, \$ST, \$SD, \$SV, \$STZ (if present, optimize_for_search SDATA
 preference)
- DR\$index name\$S (if present, optimize for sort SDATA preference)

You can find associated indexes with a particular table by querying the <code>OBJECT_TYPE</code> column of the <code>ALL OBJECTS</code> data dictionary view. To find a list of internal Oracle Text tables, query the



 ${\tt USER_TABLES} \ \textbf{table}, \ \textbf{in the schema where index was created}. \ \textbf{For example}, \ \textbf{for an index named} \\ {\tt my index:}$

```
SELECT TABLE_NAME FROM USER_TABLES WHERE TABLE_NAME LIKE 'DR$my_index%';
```

2.15.2 Configuring an Oracle Text DR\$ Index Tables Audit Policy

You can use the ACTIONS clause in the CREATE AUDIT POLICY statement to create a unified audit policy on Oracle Text DR\$ index tables.

Use the following syntax to create a unified audit policy for a table that has an Oracle Text
 DR\$ index table:

```
CREATE AUDIT POLICY policy_name
ACTIONS action ON schema.table, action ON schema.DR$index table;
```

For example, to audit the main index table (using the \$I keyword) for a table (sales) that has an index named sales idx:

```
CREATE AUDIT POLICY sales_pol ACTIONS ALL ON sales, ALL ON DR$sales idx$1;
```

2.15.3 Example: Auditing Update Actions on an Oracle Text DR\$ Index Table

The CREATE AUDIT POLICY statement can audit all or specific actions on an Oracle Text DR\$ index table.

Example 2-8 shows how to create and enable a unified audit policy for the <code>emp_data</code> table that captures user update attempts on this table's Oracle Text index table, <code>DR\$emp_data_idx\$I</code>.

Example 2-8 Auditing Update Actions on an Oracle Text DR\$ Index Table

```
CREATE AUDIT POLICY emp_data_pol ACTIONS UPDATE ON emp_data,
UPDATE ON DR$emp_data_idx$1;

AUDIT POLICY emp data pol;
```

2.15.4 How Oracle Text DR\$ Index Table Entries Appear in the Audit Trail

The <code>UNIFIED_AUDIT_TRAIL</code> data dictionary view lists actions on audited Oracle Text <code>DR\$</code> index tables.

For example:



Oracle Text CONTAINS Query Operators

This chapter describes operator precedence and provides descriptions, syntax, and examples for every CONTAINS operator.

This chapter contains the following topics:

- Operator Precedence
- ABOUT
- ACCUMulate (_)
- AND (&)
- Broader Term (BT_ BTG_ BTP_ BTI)
- CTXFILTERCACHE
- DEFINEMERGE
- DEFINESCORE
- EQUIValence (=)
- Fuzzy
- HASPATH
- INPATH
- MDATA
- MINUS (-)
- MNOT
- Narrower Term (NT_ NTG_ NTP_ NTI)
- NDATA
- NEAR (;)
- NEAR2
- NOT (~)
- OR (|)
- Preferred Term (PT)
- Related Term (RT)
- SDATA
- soundex (!)
- stem (\$)
- Stored Query Expression (SQE)
- SYNonym (SYN)
- threshold (>)
- Translation Term (TR)

- Translation Term Synonym (TRSYN)
- Top Term (TT)
- weight (*)
- wildcards (%)
- WITHIN
- Supported Oracle Text CONTAINS Query Operators for In-Memory Full Text Search

3.1 Operator Precedence

Operator precedence determines the order in which the components of a query expression are evaluated. Text query operators can be divided into two sets of operators that have their own order of evaluation. These two groups are described later as Group 1 and Group 2.

In all cases, query expressions are evaluated in order from left to right according to the precedence of their operators. Operators with higher precedence are applied first. Operators of equal precedence are applied in order of their appearance in the expression from left to right.

- Group 1 Operators
- Group 2 Operators and Characters
- Procedural Operators
- Precedence Examples
- Altering Precedence

3.1.1 Group 1 Operators

Within query expressions, the Group 1 operators have the following order of evaluation from highest precedence to lowest:

- 1. EQUIValence (=)
- 2. NEAR (;)
- 3. weight (*), threshold (>)
- 4. MINUS (-)
- 5. NOT (~)
- 6. MNOT
- 7. WITHIN
- 8. AND (&)
- 9. OR (|)
- 10. ACCUMulate ()

3.1.2 Group 2 Operators and Characters

Within query expressions, the Group 2 operators have the following order of evaluation from highest to lowest:

- 1. Wildcard Characters
- 2. stem (\$)



- **3.** Fuzzy
- 4. soundex (!)

3.1.3 Procedural Operators

Other operators not listed under Group 1 or Group 2 are procedural. These operators have no sense of precedence attached to them. They include the SQE and thesaurus operators.

3.1.4 Precedence Examples

Table 3-1 Query Expression Precedence Examples

Query Expression	Order of Evaluation
w1 w2 & w3	(w1) (w2 & w3)
w1 & w2 w3	(w1 & w2) w3
?w1, w2 w3 & w4	(?w1), (w2 (w3 & w4))
abc = def ghi & jkl = mno	((abc = def) ghi) & (jkl=mno)
dog and cat WITHIN body	dog and (cat WITHIN body)

In the first example, because AND has a higher precedence than OR, the query returns all documents that contain w1 and all documents that contain both w2 and w3.

In the second example, the query returns all documents that contain both w1 and w2 and all documents that contain w3.

In the third example, the fuzzy operator is first applied to w1, then the AND operator is applied to arguments w3 and w4, then the OR operator is applied to term w2 and the results of the AND operation, and finally, the score from the fuzzy operation on w1 is added to the score from the OR operation.

The fourth example shows that the equivalence operator has higher precedence than the ${\tt AND}$ operator.

The fifth example shows that the AND operator has lower precedence than the WITHIN operator.

3.1.5 Altering Precedence

Precedence is altered by grouping characters as follows:

- Within parentheses, expansion or execution of operations is resolved before other expansions regardless of operator precedence.
- Within parentheses, precedence of operators is maintained during evaluation of expressions.
- Within parentheses, expansion operators are not applied to expressions unless the operators are also within the parentheses.



"Grouping Characters" in Special Characters in Oracle Text Queries



3.2 ABOUT

Use the ABOUT operator to return documents that are related to a query term or phrase.

General Behavior

In English and French, ABOUT enables you to query on concepts, even if a concept is not actually part of a query. For example, an ABOUT query on heat might return documents related to temperature, even though the term *temperature* is not part of the query.

In other languages, using ABOUT will often increase the number of returned documents and may improve the sorting order of results. For all languages, Oracle Text scores results for an ABOUT query with the most relevant document receiving the highest score.

English and French Behavior

In English and French, use the ABOUT operator to query on concepts. The system looks up concept information in the theme component of the index. Create a theme component to your index by setting the INDEX THEMES BASIC_LEXER attribute to YES.



You need not have a theme component in the index to enter ${\tt ABOUT}$ queries in English and French. However, having a theme component in the index yields the best results for ${\tt ABOUT}$ queries.

Oracle Text retrieves documents that contain concepts that are related to your query word or phrase. For example, if you enter an ABOUT query on *California*, the system might return documents that contain the terms *Los Angeles* and *San Francisco*, which are cities in California. The document need not contain the term *California* to be returned in this ABOUT query.

The word or phrase specified in your ABOUT query need not exactly match the themes stored in the index. Oracle Text normalizes the word or phrase before performing lookup in the index.

You can use the ABOUT operator with the CONTAINS and CATSEARCH SQL operators. In the case of CATSEARCH, you must use query templating with the CONTEXT grammar to query on the indexed themes. See ABOUT Query with CATSEARCH in the Examples section.



Syntax

Syntax	Description
about(phrase)	In all languages, increases the number of relevant documents returned for the same query without the ABOUT operator. The <i>phrase</i> parameter can be a single word or a phrase, or a string of words in free text format.
	In English and French, returns documents that contain concepts related to phrase, provided the BASIC_LEXER INDEX_THEMES attribute is set to YES at index time.
	The score returned is a relevance score.
	Oracle Text ignores any query operators that are included in phrase.
	If your index contains only theme information, an ABOUT operator and operand must be included in your query on the text column or else Oracle Text returns an error.
	The phrase you specify cannot be more than 4000 characters.

Case-Sensitivity

ABOUT queries give the best results when your query is formulated with proper case. This is because the normalization of your query is based on the knowledge catalog which is case-sensitive.

However, you need not type your query in exact case to obtain results from an ABOUT query. The system does its best to interpret your query. For example, if you enter a query of *CISCO* and the system does not find this in the knowledge catalog, the system might use *Cisco* as a related concept for look-up.

Improving ABOUT Results

The ABOUT operator uses the supplied knowledge base in English and French to interpret the phrase you enter. Your ABOUT query therefore is limited to knowing and interpreting the concepts in the knowledge base.

Improve the results of your ABOUT queries by adding your application-specific terminology to the knowledge base.



"Extending the Knowledge Base" in Oracle Text Utilities

Limitations

- The phrase you specify in an ABOUT query cannot be more than 4000 characters.
- The ABOUT query operator is not supported within sections.
- The JSON TEXTCONTAINS query does not support the ABOUT operator.

Examples for ABOUT Operator

Single Words

To search for documents that are about soccer, use the following syntax:



```
'about (soccer) '
```

Phrases

Further refine the query to include documents about soccer rules in international competition by entering the phrase as the query term:

```
'about (soccer rules in international competition)'
```

In this English example, Oracle Text returns all documents that have themes of soccer, rules, or international competition.

In terms of scoring, documents which have all three themes will generally score higher than documents that have only one or two of the themes.

Unstructured Phrases

You can also query on unstructured phrases, such as the following:

```
'about (japanese banking investments in indonesia)'
```

Combined Queries

Use other operators, such as AND or NOT, to combine ABOUT queries with word queries. For example, enter the following combined ABOUT and word query:

```
'about (dogs) and cat'
```

Combine an ABOUT guery with another ABOUT guery as follows:

```
'about (dogs) not about (labradors) '
```



You cannot combine ABOUT with the WITHIN operator, as for example 'ABOUT (xyz) WITHIN abc'.

ABOUT Query with CATSEARCH

Enter ABOUT queries with CATSEARCH using the query template method with grammar set to CONTEXT as follows:

3.3 ACCUMulate (,)

Use the ACCUM operator to search for documents that contain at least one occurrence of any query terms, with the returned documents ranked by a cumulative score based on how many query terms are found (and how frequently).

Syntax

Syntax	Description
term1,term2 term1 ACCUM term2	Returns documents that contain <i>term1</i> or <i>term2</i> . Ranks documents according to document term weight, with the highest scores assigned to documents that have the highest total term weight.

ACCUMulate Scoring

ACCUMulate first scores documents on how many query terms a document matches. A document that matches more terms will always score higher than a document that matches fewer terms, even if the terms appear more frequently in the latter. In other words, if you search for *dog ACCUM cat*, you'll find that

the dog played with the cat

scores higher than

the big dog played with the little dog while a third dog ate the dog food

Scores are divided into ranges. In a two-term ACCUM, hits that match both terms will always score between 51 and 100, whereas hits matching only one of the terms will score between 1 and 50. Likewise, for a three-term ACCUM, a hit matching one term will score between 1 and 33; a hit matching two terms will score between 34 and 66, and a hit matching all three terms will score between 67 and 100. Within these ranges, normal scoring algorithms apply.



The Oracle Text Scoring Algorithm for more information on how scores are calculated

You can assign different weights to different terms. For example, in a query of the form soccer, Brazil*3

the term Brazil is weighted three times as heavily as soccer. Therefore, the document

people play soccer because soccer is challenging and fun

will score lower than

Brazil is the largest nation in South America

but both documents will rank below

soccer is the national sport of Brazil

Note that a query of soccer ACCUM Brazil*3 is equivalent to soccer ACCUM Brazil ACCUM Brazil ACCUM Brazil. Because each query term Brazil is considered independent, the entire query is scored as though it has four terms, not two, and thus has four scoring ranges. The first Brazil-and-soccer example document shown above scores in the first range (1-25), the second scores in the third range (51-75), and the third scores in the fourth range (76-100). (No document scores in the second range, because any document with Brazil in it will be considered to match at least three query terms.)



Example for ACCUM Operator

```
set serveroutput on;
DROP TABLE accumtbl;
CREATE TABLE accumtbl (id NUMBER, text VARCHAR2(4000));
INSERT INTO accumtbl VALUES ( 1, 'the little dog played with the big dog while the other dog ate the dog food');
INSERT INTO accumtbl values (2, 'the cat played with the dog');
CREATE INDEX accumtbl_idx ON accumtbl (text) indextype is ctxsys.context;
PROMPT dog ACCUM cat
SELECT SCORE(10) FROM accumtbl WHERE CONTAINS (text, 'dog ACCUM cat', 10) > 0;
PROMPT dog*3 ACCUM cat
SELECT SCORE(10) FROM accumtbl WHERE CONTAINS (text, 'dog*3 ACCUM cat', 10) > 0:
```

This produces the following output. Note that the document with both *dog* and *cat* scores highest.

Related Topics

weight (*)

3.4 AND (&)

Use the ${\tt AND}$ operator to search for documents that contain at least one occurrence of each of the query terms.

The AND operator returns documents that contain *all* of the query terms, while OR operator returns documents that contain *any* of the query terms.

Syntax

Syntax	Description
term1&term2	Returns documents that contain term1 and term2. Returns the minimum score of
term1 and term2	its operands. All query terms must occur; lower score taken.

Example for AND Operator

To obtain all the documents that contain the terms *blue* and *green* and *red*, enter the following query:



'blue & green & red'

In an AND query, the score returned is the score of the lowest query term. In this example, if the three individual scores for the terms *blue*, *green*, and *red* is 10, 20 and 30 within a document, the document scores 10.

Related Topics

OR (|)

3.5 Broader Term (BT, BTG, BTP, BTI)

Use the broader term operators (BT, BTG, BTP, BTI) to expand a query to include the term that has been defined in a thesaurus as the broader or higher level term for a specified term. They can also expand the query to include the broader term for the broader term and the broader term for that broader term, and so on up through the thesaurus hierarchy.

Syntax

Syntax	Description	
BT(term[(qualifier)][,level][,thes])	Expands a query to include the term defined in the thesaurus as a broader term for term.	
BTG(term[(qualifier)][,level][,thes])	Expands a query to include all terms defined in the thesaurus as broader generic terms for term.	
BTP(term[(qualifier)][,level][,thes])	Expands a query to include all the terms defined in the thesaurus as broader partitive terms for term.	
BTI(term[(qualifier)][,level][,thes])	Expands a query to include all the terms defined in the thesaurus as broader instance terms for term.	

term

Specify the operand for the broader term operator. Oracle Text expands term to include the broader term entries defined for the term in the thesaurus specified by thes. For example, if you specify BTG(dog), the expansion includes only those terms that are defined as broader term generic for dog. You cannot specify expansion operators in the term argument. The number of broader terms included in the expansion is determined by the value for level.

qualifier

Specify a qualifier for term, if term is a homograph (word or phrase with multiple meanings, but the same spelling) that appears in two or more nodes in the same hierarchy branch of thes.

If a qualifier is not specified for a homograph in a broader term query, the query expands to include the broader terms of all the homographic terms.

level

Specify the number of levels traversed in the thesaurus hierarchy to return the broader terms for the specified term. For example, a level of 1 in a BT query returns the broader term entry, if one exists, for the specified term. A level of 2 returns the broader term entry for the specified term, as well as the broader term entry, if one exists, for the broader term.

The level argument is optional and has a default value of one (1). Zero or negative values for the level argument return only the original query term.



thes

Specify the name of the thesaurus used to return the expansions for the specified term. The thes argument is optional and has a default value of DEFAULT. A thesaurus named DEFAULT must exist in the thesaurus tables if you use this default value.



If you specify thes, then you must also specify level.

Examples for Broader Term Operators

The following query returns all documents that contain the term *tutorial* or the BT term defined for *tutorial* in the DEFAULT thesaurus:

```
'BT(tutorial)'
```

When you specify a thesaurus name, you must also specify level as in:

```
'BT(tutorial, 2, mythes)'
```

Broader Term Operator on Homographs

If machine is a broader term for crane (building equipment) and bird is a broader term for crane (waterfowl) and no qualifier is specified for a broader term query, the query

```
BT (crane)
```

expands to:

```
'{crane} or {machine} or {bird}'
```

If waterfowl is specified as a qualifier for crane in a broader term query, the query

```
BT(crane{(waterfowl)})
```

expands to the query:

```
'{crane} or {bird}'
```



When specifying a qualifier in a broader or narrower term query, the qualifier and its notation (parentheses) must be escaped, as is shown in this example.

Related Topics

CTX_THES.BT in CTX_THES Package for more information on browsing the broader terms in your thesaurus

3.6 CTXFILTERCACHE

Oracle Text provides a cache layer called query filter cache that can be used to cache the query results. Query filter cache is sharable across queries. Thus, the cached query results can be reused by multiple queries, improving the query response time. The CTXFILTERCACHE

operator is used to specify which query results or part of query results to cache in the query filter cache.

CTXFILTERCACHE only supports CONTEXT grammar queries. CONTAINER queries like template queries are not supported. If you execute it with a template query, then errors are raised.



The CTXFILTERCACHE query operator was designed to speed up commonly-used expressions in queries. In Oracle Database Release 21c, this function is replaced by other internal improvements. The CTXFILTERCACHE operator is deprecated (and will pass through its operands to be run as a normal query). Because they no longer have a function, the view CTX_FILTER_CACHE_STATISTICS is also deprecated, and also the storage attribute QUERY FILTER CACHE SIZE.

A

Caution:

Before using CTXFILTERCACHE, you must run PURGE recyclebin as follows:

```
SQL> PURGE recylebin;
```

See *Oracle Database Administrator's Guide* for complete information about purging objects in the recycle bin.

Syntax

```
ctxfiltercache((query text) [, save score] [, topN])
```

query_text

Specify the query whose results need to be stored in the cache.

save_score

Specify TRUE if you want to cache all the query results along with their scores in the cache. The default is FALSE. In this case, a score of 100 is returned for each query result, and these scores are not stored in the cache. Only the query results are stored in the cache. Specify FALSE when you want to reuse the query results and not their scores in other queries. This is particularly useful when you use the query text as a filter, such as a security filter, where the relevance of the cached part of the query does not affect the relevance of the query as a whole. Thus, when used with the AND operator (which returns a lower score of its operands), a score of 100 does not affect the score of a query as a whole.

topN

Specify TRUE if you want only the highest scoring query results to be stored in the cache. Oracle Text internally determines how many highest scoring query results to store in the cache. This helps in reducing the memory consumption of the cache.





If you specify TRUE for topN, then save score should also be TRUE.

Examples for CTXFILTERCACHE

Stored Query Results and TopN Examples

The following example stores the query results of the common predicate query in the cache:

```
select * from docs where contains(txt, 'ctxfiltercache((common predicate), FALSE)')>0;
```

Here, save_score is FALSE, and hence the score of 100 is returned for each query result, and the scores are not stored in the cache.

In the following example, the cached results of the <code>common_predicate</code> query are reused by the <code>new query</code> query.

```
select * from docs where contains(txt, 'new_query & ctxfiltercache((common_predicate),
FALSE)')>0;
```

Set save_score to TRUE as shown in the following example to store all the query results of the common predicate query, along with the actual scores, in the cache.

```
select * from docs where contains(txt, 'ctxfiltercache((common predicate), TRUE)')>0;
```

Set topN to TRUE if you want to store only the highest scoring query results of the common predicate query in the cache as described in the following example.

```
select id, score(1) from docs where contains(txt, 'ctxfiltercache((common_predicate),
TRUE, TRUE)', 1)>0 order by score(1) desc;
```

Set topN to TRUE for the main part of the query and FALSE for the filter part, when the score is relevant only for the main part of the query. The following example shows a query with two ctxfiltercache clauses. It performs a free-text search for "cat AND dog" and then applies a security filter to the search operation. Results of both the parts of this query are separately cached so that they can be reused, but the score is relevant only for the first part of the query.

```
select id, score(1) from docs where contains(txt, 'ctxfiltercache((cat AND dog), TRUE, TRUE) AND ctxfiltercache((john WITHIN allowedUsers), FALSE, FALSE)', 1) > 0;
```

Cached Score Example

CTXFILTERCACHE stores one query result for score at a time in the cache. Hence, two similar queries executed serially are considered the same query, and there is only one such query stored in the cache.

The following examples, query *A* and query *B*, show two similar queries. The hit score for *A* is 100, and the hit score for *B* is 5. Assume the cache is empty and you execute query *A* first. The computed score 100 is stored in the cache for this query. When you execute query *B* subsequently now, the cache contains the stored score of 100, and therefore, query *B* returns the cached score of 100. Conversely, if you execute query *B* before query *A*, then the cached computed score that gets returned is 5.

Query A:



```
,OR,ADD
)),T,T)',1)>0 order by score(1) desc;
```

Query B:

Notes

The guery filter cache is an index level storage preference.

The storage preference for the query filter cache can be set at *partition* level only if this is also set at *index* level. If a filter cache preference is set at partition level without any filter cache preference being set at index level, then an error is thrown as follows: "Illegal syntax for index, preference, source or section name."

Note that CTXFILTERCACHE is not utilized with:

- INPATH/HASPATH queries
- First query after syncindex for NDATA and SDATA

When topn is false, the CTXFILTERCACHE operator can be either a top-level operator or a child of the following operators:

- AND
- ACCUM
- NOT
- OR
- THRESHOLD (left side operand only)
- WEIGHT (left side operand only)

When topN is TRUE:

- The ctxfiltercache operator can be either a top-level operator or a child of the following operators:
 - AND
 - THRESHOLD (left side operand only)
 - WEIGHT (left side operand only)
- TopN is enabled only when the ctxfiltercache operator is used with the order key
 ORDER BY SCORE(n) DESC and Oracle hint DOMAIN_INDEX_SORT for global index.
 Additionally, for a partitioned index, be sure to have partition pruning in your query.
 Otherwise, although topN is set to TRUE, normal mode will be used instead of topN mode.



Note:

The ctxfiltercache operator only supports a CONTEXT grammar query. This means that container queries like template queries are not supported.

If ctxfiltercache is used with a query template, then the following type of error will occur:

```
ERROR at line 1:
ORA-29902: error in executing ODCIIndexStart() routine
ORA-20000: Oracle Text error:
DRG-50900: text query parser error on line 1, column 8
DRG-50905: invalid score threshold <textquery
```

An example of a query that results in this error is as follows:

```
select score(1), id, txt from tdrbqfc45 where contains(txt,
'ctxfiltercache((<query><textquery>near2((a,b,c,d))
</textquery><score datatype="FLOAT"/>
</query>),true)', 1)>0 order by id;
```

To use <code>ctxfiltercache</code> you must specify a size for the query filter cache using the basic storage attribute <code>query_filter_cache_size</code>. The default size is <code>0</code>, which means that <code>ctxfiltercache</code> is disabled by default.

The view ctx_filter_cache_statistics provides various statistics about the query filter cache.

The query filter cache does not differentiate queries that only vary in how the score is computed. Score is never computed on the fly within the query filter cache. See "Cached Score Example" for an illustration of how this works.

Note:

Direct functional evaluation for \mathtt{CTXCAT} index is not supported. To achieve functional evaluation, you must add a hint in the query as follows :

```
\label{lem:continuous} $$ select /*+ index(tkctobcr11_12_2 tkctobcr11_12_2x_title) */ * from tkctobcr11_12_2 $$ where CATSEARCH(title,'pokemon','category_id=9')>0 and contains(clb,'SQL,sdata(id between 1 and 1000)')>0 order by id;
```

Related Topics

"CTX_FILTER_CACHE_STATISTICS" for more information about the ctx filter cache statistics view

"BASIC_STORAGE" for more information about the query_filter_cache_size basic storage attribute

3.7 DEFINEMERGE

Use the DEFINEMERGE operator to define how the score of child nodes of the AND and OR should be merged.

The DEFINEMERGE operator can be used as operand(s) of any operators that allow AND or OR as operands. The score can be merged in three ways: picking the minimum value, picking the maximum value, or calculating the average score of all child nodes.

Use **DEFINESCORE** before using **DEFINEMERGE**.

Syntax

```
DEFINEMERGE ( ( (text_query1), (text_query2), ... ) , operator, merge_method )
```

Syntax	Description
text_query1,2	Defines the search criteria. These parameters can have any value that is valid for the AND/OR operator.
operator	Defines the relationship between the two text_query parameters.
merge_method	Defines how the score of the text_query should be merged. Possible values: ${\tt MIN},$ ${\tt MAX},$ ${\tt AVG},$ ${\tt ADD}$

Example for DEFINEMERGE Operator

```
'DEFINEMERGE (((dog , cat) , (blue or green)), AND, MIN )'
```

Queries for the expression "dog ACCUM cat" and "blue OR green," using the default scoring schemes and then using the minimum score of the two as the merged-score.

```
'DEFINEMERGE ( ((DEFINESCORE (dog, DISCRETE)) , (cat)), AND, MAX)'
```

Queries for the term "dog" using the DISCRETE scoring, and for the term "cat" using the default relevant scoring, and then using the maximum score of the two as the merged-score.

Example 3-1 DEFINEMERGE and text_query

The following examples show only the text query part of a CONTAINS query:

```
'DEFINEMERGE ( ((dog), (cat)), OR, AVG)'
```

Queries for the term "dog" or "cat," using the average relevance score of both terms as the merged score.

Related Topic

DEFINESCORE.

3.8 DEFINESCORE

Use the DEFINESCORE operator to define how a term or phrase, or a set of term equivalences will be scored. The definition of a scoring expression can consist of an arithmetic expression of predefined scoring components and numeric literals.

DEFINEMERGE can be used after DEFINESCORE.

Syntax

DEFINESCORE (query_term, scoring_expression)

query_term

The query term or phrase. Expressions containing the following operators are also allowed:

Operators	Operators
-	-
ABOUT	EQUIV(=)
Fuzzy	Soundex (!)
Stem (\$)	Wildcards (% _)
SDATA	MDATA

scoring_expression

An arithmetic expression that describes how the <code>query_term</code> should be scored. This operand is a string that contains the following components:

- Arithmetic operators: + * /. The precedence is multiplication and division (*, /) first before addition and subtraction (+, -).
- Grouping operators: (). Parentheses can be used to alter the precedence of the arithmetic operators.
- Absolute function: ABS (n) returns the absolute value of n; where n is any expression that returns a number.
- Logarithmic function: LOG(n) returns the base-10 logarithmic value of n; where n is any expression that returns a number.
- Predefined scoring components: Each of the following scoring components returns a value of 0 - 100, depending on different criteria:

Name	Description
DISCRETE	If the term exists in the document, score = 100. Otherwise, score = 0.
OCCURRENCE	Score based on the number of occurrences.
RELEVANCE	Score based on the document's relevance.
COMPLETION	Score based on coverage. Documents will score higher if the ratio between the number of the matching terms and the number of all terms in the section (counting stop words) is higher. The COMPLETION scoring is only applicable when used with the WITHIN operator to search in zone sections.
IGNORE	Ignore the scoring of this term. This component should be used alone. Otherwise, the query will return a syntax error. If the scoring of the only term in the query is set to IGNORE, then all the matching documents should be returned with the same score of 100.





For numeric literals, any number literal can be used that conforms to the SQL pattern of number literal, and is within the range of the double precision floating point (-3.4e38 to 3.4e38).

scoring_expression Syntax

Examples for DEFINESCORE Operator

```
'DEFINESCORE (dog, OCCURRENCE)'
```

Queries for the word *dog*, and scores each document using the occurrence score. Returns the score as integer.

```
'DEFINESCORE (Labradors are big dog, RELEVANCE)'
```

Queries for the phrase *Labradors are big dogs*, and scores each document using the relevance score.

```
'cat and DEFINESCORE (dog, IGNORE)'
```

Queries for the words *dog* and *cat*, using only the default relevance score of *cat* as the overall score of the document. Returns the score as integer.

```
'DEFINESCORE (dog, IGNORE)'
```

Queries for the word *dog*, and returns all documents with the word *dog*. The result is the same as if all documents get a score of 100. Returns the score as integer.

```
'DEFINESCORE (dog, ABS (100-RELEVANCE))'
```

Queries for the word *dog*, and scores each document using the absolute value of 100 minus the relevance score. Returns the score as integer.

```
'cat and DEFINESCORE (dog, RELEVANCE*5 - OCCURRENCE)'
```

Returns a syntax error: Two predefined components are used.

When DEFINESCORE is used with query templates, the scoring_expression overrides the values specified by the template. The following example queries for "dog" and "cat," scores "cat" using OCCURRENCE (COUNT) and scores "dog" based on RELEVANCE.



Limitations

- If the ABOUT operator is used in query_term, the OCCURRENCE and COMPLETION scoring will not be applicable. If used, the query will return a syntax error.
- The IGNORE score cannot be used as right hand of the minus operator. If used, then a syntax error will occur.
- The COMPLETION score is only applicable if the DEFINESCORE is used with a WITHIN operator to search in zone sections, for example:

```
'DEFINESCORE (dog, COMPLETION) within zonesection'
```

otherwise, the query will return a syntax error.

- For the left hand operand of WITHIN:
 - All nodes must use the same predefined-scoring component. (If not specified, then the predefined scoring is RELEVANCE.)
 - If the nodes use DISCRETE or COMPLETION, then only the AND and OR operator is allowed
 as the left hand children of WITHIN.
 - If the nodes use DISCRETE or COMPLETION, then WITHIN will use the max score of all section instances as the score.
 - If the nodes use RELEVANCE or OCCURRENCE, then WITHIN will use the summation of the score of all section instances as the score.
- Only one predefined scoring component can be used in the scoring_expression at one
 time. If more than one predefined scoring component is used, then a syntax error will
 occur.



Oracle Database SQL Language Reference

Notes

- The DEFINESCORE operator, the absolute function, the logarithmic function, and the predefined scoring components are case-insensitive.
- The query term and the scoring expression parameters are mandatory.
- The final score of the DEFINESCORE operator will be truncated to be in the 0 100 range. If the data type is INTEGER, then the score is rounded up.
- The intermediate data type of the scoring value is a double precision float. As a result, the value is limited to be in the -3.4e38 to 3.4e38 range. If the intermediate scoring of any document exceeds the value, then the score will be truncated. If an integer scoring is required, then the score will always be rounded up after the score is calculated.
- The DEFINESCORE operator can be used as an operand of the following operators:
 - AND
 - NOT
 - INPATH
 - THRESHOLD



- WITHIN
- SQE
- OR
- DEFINEMERGE
- MINUS
- WEIGHT
- ACCUM

For example, the following statement is valid:

```
DEFINESCORE('dog', OCCURRENCE) AND DEFINESCORE('cat', RELEVANCE)
```

Queries for the term "dog" using occurrence scoring, and the term "cat" using relevance scoring.

• If DEFINESCORE is used as a parameter of other operators, then an error will be returned. For example, the following example returns an error:

```
SYN(DEFINESCORE('cat', OCCURRENCE))
```

 When used with query templates, the scoring_expression overrides the values specified by the template. For example,

Queries for "dog" and "cat", scores "cat" using <code>OCCURRENCE(COUNT)</code>, and scores "dog" based on <code>RELEVANCE</code>.

Related Topic

DEFINEMERGE.

3.9 EQUIValence (=)

Use the EQUIV operator to specify an acceptable substitution for a word in a query.

Syntax

Syntax	Description
term1=term2	Specifies that term2 is an acceptable substitution for term1. Score calculated
term1 equiv term2	as the sum of all occurrences of both terms.

Example for EQUIV Operator

The following example returns all documents that contain either the phrase *alsatians* are big dogs or *labradors* are big dogs:



^{&#}x27;labradors=alsatians are big dogs'

Operator Precedence

The EQUIV operator has higher precedence than all other operators except the expansion operators (fuzzy, soundex, stem).

3.10 Fuzzy

Use the \mathtt{fuzzy} operator to expand queries to include words that are spelled similarly to the specified term. This type of expansion is helpful for finding more accurate results when there are frequent misspellings in your document set.

The fuzzy syntax enables you to rank the result set so that documents that contain words with high similarity to the query word are scored higher than documents with lower similarity. You can also limit the number of expanded terms.

Unlike stem expansion, the number of words generated by a fuzzy expansion depends on what is in the index. Results can vary significantly according to the contents of the index.

Supported Languages

Oracle Text supports fuzzy definitions for English, French, German, Italian, Dutch, Spanish, Portuguese, Japanese, OCR, and auto-language detection.

Stopwords

If the fuzzy expansion returns a stopword, the stopword is not included in the query or highlighted by CTX DOC.HIGHLIGHT OR CTX DOC.MARKUP.

Base-Letter Conversion

If base-letter conversion is enabled for a text column and the query expression contains a fuzzy operator, Oracle Text operates on the base-letter form of the query.

Syntax

fuzzy(term, score, numresults, weight)

Description
Specify the word on which to perform the fuzzy expansion. Oracle Text expands term to include words only in the index. The word needs to be at least 3 characters for the fuzzy operator to process it.
Specify a similarity score. Terms in the expansion that score below this number are discarded. Use a number between 1 and 80. The default is 60.
Specify the maximum number of terms to use in the expansion of term. Use a number between 1 and 5000. The default is 100.
Specify WEIGHT or W for the results to be weighted according to their similarity scores. Specify NOWEIGHT or N for no weighting of results.

Examples for Fuzzy Operator

Consider the CONTAINS query:

```
...CONTAINS(TEXT, 'fuzzy(government, 70, 6, weight)', 1) > 0;
```



This query expands to the first six fuzzy variations of *government* in the index that have a similarity score over 70.

In addition, documents in the result set are weighted according to their similarity to *government*. Documents containing words most similar to government receive the highest score.

Skip unnecessary parameters using the appropriate number of commas. For example:

Backward Compatibility Syntax

The old fuzzy syntax from previous releases is still supported. This syntax is as follows:

Parameter	Description
?term	Expands term to include all terms with similar spellings as the specified term. Term needs to be at least 3 characters for the fuzzy operator to process it.

3.11 HASPATH

Use the HASPATH operator to find all XML documents that contain a specified section path. You can also use this operator to do section equality testing.

Your index must be created with the PATH SECTION GROUP for this operator to work.

Syntax

Syntax	Description
HASPATH(path)	Searches an XML document set and returns a score of 100 for all documents where <i>path</i> exists. Separate parent and child paths with the / character. For example, you can specify <i>A/B/C</i> .
	See example.
HASPATH(A="value")	Searches an XML document set and returns a score of 100 for all documents that have the element A with content <i>value</i> and only <i>value</i> .
	See example.

Using Special Characters with HASPATH and INPATH

The following rules govern the use of special characters with regard to both the HASPATH and INPATH operators:

- Left-brace ({) and right-brace (}) characters are not allowed inside HASPATH or INPATH expressions unless they are inside the equality operand enclosed by double quotes. So both 'HASPATH({/A/B})' and 'HASPATH(/A/{B})' will return errors. However, 'HASPATH(/A[B="{author}])' will be parsed correctly.
- With exception of the backslash (\), special characters, such as dollar sign (\$), percent sign (%), underscore (_), left brace ({}), and right brace ({}), when inside the equality operand enclosed by double or single quotes, have no special meaning. (That is, no stemming, wildcard expansion, or similar processing will be performed on them.) However, they are still subject to regular text lexing and will be translated to whitespace, with the exception of



^{&#}x27;fuzzy(government,,,weight)'

characters declared as printjoins. A backslash will still escape any character that immediately follows it.

For example, if the hyphen (-) and the double quote character (") are defined as printjoins in a lexer preference, then:

- The string B_TEXT inside HASPATH (/A[B="B_TEXT") will be lexed as the phrase B
 TEXT.
- The string B-TEXT inside HASPATH (/A[B="B-TEXT") will be lexed as the word B-TEXT.
- The string B'TEXT inside HASPATH (/A[B="B'TEXT") will be lexed as the word B"TEXT.
 You must use a backslash to escape the double quote between B and TEXT, or you will get a parsing error.
- The string {B_TEXT} inside HASPATH (/A[B="{B_TEXT}") will be lexed as a phrase B
 TEXT.

Examples for HASPATH Operator

Path Testing

The query

HASPATH (A/B/C)

finds and returns a score of 100 for the document

<A><C>dog</C>

without the query having to reference dog at all.

Section Equality Testing

The query

dog INPATH A

finds

<A>dog

but it also finds

<A>dog park

To limit the query to the term *dog* and nothing else, you can use a section equality test with the HASPATH operator. For example,

HASPATH (A="dog")

finds and returns a score of 100 only for the first document, and not the second.

Limitations

Because of how XML section data is recorded, false matches might occur with XML sections that are completely empty as follows:

```
<A><B><C></C></B><D><E></E></D></A>
```

A query of HASPATH (A/B/E) or HASPATH (A/D/C) falsely matches this document. This type of false matching can be avoided by inserting text between empty tags.



False matches might also occur when the document has empty elements but has values in attributes, as in the following example document:

```
<Test>
<Client id="1">
<Info infoid="1"/>
</Client>
<Client id="2">
<Info infoid="2"/>
</Client>
</Client>
```

When searching with the following query, the query returns the document shown in the example, which is a false match.

The following query was used to return the example document, which is a false match:

```
SELECT main_detail_logging_id, t.xml_data.getstringval() xml_data FROM
TEST_XMLTYPE t
WHERE CONTAINS(t.xml_data,
'HASPATH(/Test/Client[@id="1"]/Info[@infoid="2"])') > 0;
```

3.12 INPATH

Use the INPATH operator to do path searching in XML documents. This operator is like the ${\tt WITHIN}$ operator except that the right-hand side is a parentheses enclosed path, rather than a single section name.

Your index must be created with the PATH SECTION GROUP for the INPATH operator to work.

Syntax

The ${\tt INPATH}$ operator has the following syntax:

Top-Level Tag Searching

Syntax	Description
term INPATH (/A)	Returns documents that have term within the <a> and
term INPATH (A)	tags.

Any-Level Tag Searching

Syntax	Description
term INPATH (//A)	Returns documents that have <i>term</i> in the <a> tag at any level. This query is the same as <i>'term WITHIN A'</i>

Direct Parentage Path Searching

Syntax	Description
term INPATH (A/B)	Returns documents where <i>term</i> appears in a B element which is a direct child of a top-level A element.
	For example, a document containing
	<a>term
	is returned.



Single-Level Wildcard Searching

Syntax	Description
term INPATH (A/*/B)	Returns documents where <i>term</i> appears in a B element which is a grandchild (two levels down) of a top-level A element.
	For example a document containing
	<a><d>term</d>
	is returned.

Multi-level Wildcard Searching

Syntax	Description
term INPATH (A/*/B/*/*/C)	Returns documents where <i>term</i> appears in a C element which is 3 levels down from a B element which is two levels down (grandchild) of a top-level A element.

Any-Level Descendant Searching

Syntax	Description
term INPATH(A//B)	Returns documents where term appears in a B element which is some descendant (any level) of a top-level A element.

Attribute Searching

Syntax	Description
term INPATH (//A/@B)	Returns documents where <i>term</i> appears in the B attribute of an A element at any level. Attributes must be bound to a direct parent.

Descendant/Attribute Existence Testing

Syntax	Description
term INPATH (A[B])	Returns documents where term appears in a top-level A element which has a B element as a direct child.
term INPATH (A[.//B])	Returns documents where term appears in a top-level A element which has a B element as a descendant at any level.
term INPATH (//A[@B])	Finds documents where term appears in an A element at any level which has a B attribute. Attributes must be tied to a direct parent.

Attribute Value Testing

Syntax	Description
term INPATH (A[@B = "value"])	Finds all documents where <i>term</i> appears in a top-level A element which has a B attribute whose value is <i>value</i> .
term INPATH (A[@B != "value"])	Finds all documents where <i>term</i> appears in a top-level A element which has a B attribute whose value is not <i>value</i> .



Tag Value Testing

Syntax	Description
term INPATH (A[B = "value"]))	Returns documents where <i>term</i> appears in an A tag which has a B tag whose value is <i>value</i> .

NOT Testing

Syntax	Description
term INPATH (A[NOT(B)])	Finds documents where <i>term</i> appears in a top-level A element which does not have a B element as an immediate child.

AND and OR Testing

Syntax	Description
term INPATH (A[B and C])	Finds documents where term appears in a top-level A element which has a B and a C element as an immediate child.
term INPATH (A[B and @C="value"]])	Finds documents where <i>term</i> appears in a top-level A element which has a B element and a C attribute whose value is <i>value</i> .
term INPATH (A [B OR C])	Finds documents where <i>term</i> appears in a top-level A element which has a B element or a C element.

Combining Path and Node Tests

Syntax	Description
term INPATH (A[@B = "value"]/C/D)	Returns documents where <i>term</i> appears in aD element which is the child of a C element, which is the child of a top-level A element with a B attribute whose value is <i>value</i> .

Nested INPATH

Nest the entire INPATH expression in another INPATH expression as follows:

(dog INPATH (//A/B/C)) INPATH (D)

When you do so, the two INPATH paths are completely independent. The outer INPATH path does not change the context node of the inner INPATH path. For example:

(dog INPATH (A)) INPATH (D)

never finds any documents, because the inner INPATH is looking for dog within the top-level tag A, and the outer INPATH constrains that to document with top-level tag D. A document can have only one top-level tag, so this expression never finds any documents.

Case-Sensitivity

Tags and attribute names in path searching are case-sensitive. That is,

dog INPATH (A)



finds <A>dog but does not find <a>dog. Instead use

```
dog INPATH (a)
```

Using Special Characters with INPATH

See "Using Special Characters with HASPATH and INPATH" for information on using special characters, such as the percent sign (%) or the backslash (\), with INPATH.

Examples for INPATH Operator

Top-Level Tag Searching

To find all documents that contain the term *dog* in the top-level tag <A>:

```
dog INPATH (/A)
```

or

dog INPATH(A)

Any-Level Tag Searching

To find all documents that contain the term dog in the <A> tag at any level:

```
dog INPATH (//A)
```

This query finds the following documents:

```
<A>dog</A>
```

and

<C><A>dog</C>

Direct Parentage Searching

To find all documents that contain the term dog in a B element that is a direct child of a top-level A element:

```
dog INPATH(A/B)
```

This query finds the following XML document:

```
A>B>My dog is friendly.</B>A>
```

but does not find:

```
<C><B>My dog is friendly.</B></C>
```

Tag Value Testing

You can test the value of tags. For example, the query:

```
dog INPATH(A[B="dog"])
```

Finds the following document:

<A>dog

But does not find:

A>B>My dog is friendly.A>



Attribute Searching

You can search the content of attributes. For example, the query:

dog INPATH(//A/@B)

Finds the document

<C> </C>

Attribute Value Testing

You can test the value of attributes. For example, the query

California INPATH (//A[@B = "home address"])

Finds the document:

San Francisco, California, USA

But does not find:

San Francisco, California, USA

Path Testing

You can test if a path exists with the HASPATH operator. For example, the query:

HASPATH (A/B/C)

finds and returns a score of 100 for the document

<A><C>dog</C>

without the query having to reference dog at all.

Limitations

Testing for Equality

The following is an example of an INPATH equality test.

```
dog INPATH (A[@B = "foo"])
```

The following limitations apply for these expressions:

- Only equality and inequality are supported. Range operators and functions are not supported.
- The left hand side of the equality must be an attribute. Tags and literals here are not enabled.
- The right hand side of the equality must be a literal. Tags and attributes here are not allowed.
- The test for equality depends on your lexer settings. With the default settings, the query

```
dog INPATH (A[@B= "pot of gold"])
```

matches the following sections:

```
<A B="POT OF GOLD">dog</A>
```

and



```
<A B="pot of gold">dog</A>
```

because lexer is case-insensitive by default.

```
<A B="POT IS GOLD">dog</A>
```

because of and is are default stopwords in English, and a stopword matches any stopword word.

```
<A B="POT OF GOLD">dog</A>
```

because the underscore character is not a join character by default.

3.13 MDATA

Use the MDATA operator to query documents that contain MDATA sections. MDATA sections are metadata that have been added to documents to speed up mixed querying.

MDATA queries are treated exactly as literals. For example, with the query:

```
MDATA(price, $1.24)
```

the \$ is not interpreted as a stem operator, nor is the . (period) transformed into whitespace. A right (close) parenthesis terminates the MDATA operator, so that MDATA values that have close parentheses cannot be searched.

Syntax

```
MDATA (sectionname, value)
```

sectionname

The name of the MDATA section(s) to search. MDATA will also search DATE or numerical equality if the sectionname parameter is mapped to a FILTER BY column of DATE or some numerical type.

value

The value of the MDATA section. For example, if an MDATA section called Booktype has been created, it might have a value of *paperback*.

For MDATA operator on MDATA sections that are mapped to a DATE FILTER BY column, the MDATA value must follow the Date format: YYYY-MM-DD HH24:MI:SS. Otherwise, the expected rows will not be returned. If the time component is omitted, it will default to 00:00:00, according to SQL semantics.

Example for MDATA Operator

Suppose you want to query for books written by the writer *Nigella Lawson* that contain the word *summer*. Assuming that an MDATA section called AUTHOR has been declared, you can query as follows:

```
SELECT id FROM idx_docs
WHERE CONTAINS(text, 'summer AND MDATA(author, Nigella Lawson)')>0
```

This query will only be successful if an AUTHOR tag has the exact value *Nigella Lawson* (after simplified tokenization). *Nigella* or *Ms. Nigella Lawson* will not work.

Notes

MDATA query values ignore stopwords.



The MDATA operator returns an unlimited number of results or 0, depending on whether the document is a match. You can set the maximum.

The MDATA operator is not supported for CTXCAT and CTXRULE indexes.

Table 3-2 shows how MDATA interacts with some other query operators:

Table 3-2 MDATA and Other Query Operators

Operator	Example	Allowed?
AND	dog & MDATA(a, b)	yes
OR	dog MDATA(a, b)	yes
NOT	dog ~ MDATA(a, b)	yes
MINUS	dog - MDATA(a, b)	yes
ACCUM	dog , MDATA(a, b)	yes
PHRASE	MDATA(a, b) dog	no
NEAR	MDATA(a, b) ; dog	no
WITHIN, HASPATH, INPATH	MDATA(a, b) WITHIN c	no
Thesaurus	MDATA(a, SYN(b))	no
expansion	MDATA(a, \$b) MDATA(a, b%) MDATA(a, !b) MDATA(a, ?b)	no (syntactically allowed, but the inner operator is treated as literal text)
ABOUT	ABOUT(MDATA(a,b)) MDATA(ABOUT(a))	no (syntactically allowed, but the inner operator is treated as literal text)

When ${\tt MDATA}$ sections repeat, each instance is a separate and independent value. For instance, the document

<AUTHOR>Terry Pratchett</AUTHOR><AUTHOR>Douglas Adams</AUTHOR>

can be found with any of the following queries:

```
MDATA(author, Terry Pratchett)
MDATA(author, Douglas Adams)
MDATA(author, Terry Pratchett) and MDATA(author, Douglas Adams)
```

but not any of the following:

```
MDATA(author, Terry Pratchett Douglas Adams)
MDATA(author, Terry Pratchett & Douglas Adams)
MDATA(author, Pratchett Douglas)
```

Related Topics

"ADD_MDATA"

"ADD_MDATA_SECTION"



See Also:

Oracle Text Application Developer's Guide for information about section searching

3.14 MINUS (-)

Use the MINUS operator to lower the score of documents that contain unwanted noise terms. MINUS is useful when you want to search for documents that contain one query term but want the presence of a second term to cause a document to be ranked lower.

Syntax

Syntax	Description
term1-term2 term1 minus term2	Returns documents that contain <i>term1</i> . Calculates score by subtracting the score of <i>term2</i> from the score of <i>term1</i> . Only documents with positive score are returned.

Example for MINUS Operator

Suppose a query on the term *cars* always returned high scoring documents about *Ford cars*. You can lower the scoring of the Ford documents by using the expression:

'cars - Ford'

In essence, this expression returns documents that contain the term *cars* and possibly *Ford*. However, the score for a returned document is the score of *cars* minus the score of *Ford*.

Related Topics

"NOT (~)"

3.15 MNOT

The Mild Not (MNOT) operator is similar to the NOT and MINUS operators. The Mild Not operator returns hits where the the left child is not contained by the right child. Both children can only be TERM or PHRASE nodes.

The semantics can be illustrated with a query of "term1 mnot term1 term2", where the hits for "term1 term2" will be filtered out. For example:

- A document with only term1 will be returned, with score unchanged.
- A document with only *term1 term2* will not be returned.
- A document with term1 term1 term2 will be returned, but the score will be calculated using
 just the first term1 hit.

The behavior described in the third bullet is different from the behavior of NOT, which does not return this type of document.

The MNOT operator is more specific than the MINUS operator, in that the left child must be contained by the right child. If it is not, the Mild Not operator ignores the right child. Also, for Mild Not, the right child is a true filter, that is, it does not simply subtract the scores of left child and right child.



The MNOT operator has precedence lower than NOT and higher than WITHIN.

Syntax

Syntax	Description
term1 mnot term1 term2	Returns docs that contain <i>term1</i> unless it is part of the phrase <i>term1 term2</i> .
term1 mnot term2	Returns all documents that contain <i>term1</i> . It will be the same query as just <i>term1</i> .

Example for MNOT Operator

The children of the MNOT operator must be a TERM or PHRASE.

```
SELECT * FROM docs
WHERE CONTAINS(txt, 'term1 mnot term1 term2') >0
```

Related Topic

"NOT (~)"

3.16 Narrower Term (NT, NTG, NTP, NTI)

Use the narrower term operators (NT, NTG, NTP, NTI) to expand a query to include all the terms that have been defined in a thesaurus as the narrower or lower level terms for a specified term.

They can also expand the query to include all of the narrower terms for each narrower term, and so on down through the thesaurus hierarchy.

Syntax

Syntax	Description
NT(term[(qualifier)][,level][,thes])	Expands a query to include all the lower level terms defined in the thesaurus as narrower terms for term.
NTG(term[(qualifier)][,level][,thes])	Expands a query to include all the lower level terms defined in the thesaurus as narrower generic terms for term.
NTP(term[(qualifier)][,level][,thes])	Expands a query to include all the lower level terms defined in the thesaurus as narrower partitive terms for term.
NTI(term[(qualifier)][,level][,thes])	Expands a query to include all the lower level terms defined in the thesaurus as narrower instance terms for term.

term

Specify the operand for the narrower term operator. term is expanded to include the narrower term entries defined for the term in the thesaurus specified by thes. The number of narrower terms included in the expansion is determined by the value for level. You cannot specify expansion operators in the term argument.

qualifier

Specify a qualifier for term, if term is a homograph (word or phrase with multiple meanings, but the same spelling) that appears in two or more nodes in the same hierarchy branch of thes.

If a qualifier is not specified for a homograph in a narrower term query, the query expands to include all of the narrower terms of all homographic terms.

level

Specify the number of levels traversed in the thesaurus hierarchy to return the narrower terms for the specified term. For example, a level of 1 in an \mathbb{NT} query returns all the narrower term entries, if any exist, for the specified term. A level of 2 returns all the narrower term entries for the specified term, as well as all the narrower term entries, if any exist, for each narrower term.

The level argument is optional and has a default value of one (1). Zero or negative values for the level argument return only the original query term.

thes

Specify the name of the thesaurus used to return the expansions for the specified term. The thes argument is optional and has a default value of DEFAULT. A thesaurus named DEFAULT must exist in the thesaurus tables if you use this default value.



If you specify thes, then you must also specify level.

Examples for Narrower Term Operators

The following query returns all documents that contain either the term cat or any of the NT terms defined for cat in the DEFAULT thesaurus:

```
'NT(cat)'
```

If you specify a thesaurus name, then you must also specify level as in:

```
'NT(cat, 2, mythes)'
```

The following query returns all documents that contain either *fairy tale* or any of the narrower instance terms for *fairy tale* as defined in the DEFAULT thesaurus:

```
'NTI(fairy tale)'
```

That is, if the terms *cinderella* and *little mermaid* are defined as narrower term instances for fairy tale, Oracle Text returns documents that contain fairy tale, cinderella, or little mermaid.

Notes

Each hierarchy in a thesaurus represents a distinct, separate branch, corresponding to the four narrower term operators. In a narrower term query, Oracle Text only expands the query using the branch corresponding to the specified narrower term operator.

Related Topic

CTX_THES.NT in CTX_THES Package for more information on browsing the narrower terms in your thesaurus



3.17 NDATA

Use the ${\tt NDATA}$ operator to find matches that are spelled in a similar way or where rearranging the terms of the specified phrase is useful.

It is helpful for finding more accurate results when there are frequent misspellings (or inaccurate orderings) of name data in the document set. This operator can be used only on defined NDATA sections. The NDATA syntax enables you to rank the result set so that documents that contain words with high orthographic similarity are scored higher than documents with lower similarity.

Normalization

A lexer does not process NDATA query phrases. Users can, however, set base letter and alternate spelling attributes for a particular section group containing NDATA sections. Query case is normalized and non-character data (except for white space) is removed (for example, numerical or punctuation).

Syntax

ndata(sectionname, phrase [,order][,proximity][,threshold])

Parameter Name	Default Value	Parameter Description
sectionname		Specify the name of a defined NDATA sections to query (that is, section_name)
phrase		Specify the phrase for the name data query.
		The phrase parameter can be a single word or a phrase, or a string of words in free text format.
		The score returned is a relevant score.
		Oracle Text ignores any query operators that are included in phrase.
		The phrase should be a minimum of two characters in length and should not exceed 4000 characters in length.
order NOORDER	NOORDER	Specify whether individual tokens (terms) in a query should be matched in-order or in any order. The order parameter provides a primary filter for matching candidate documents.
		ORDER or 0 - The query terms are matched in-order.
	${\tt NOORDER}$ o ${\tt N}$ [DEFAULT] - The query terms are matched in any order.	
proximity	NOPROXIMITY	Specify whether the proximity of terms should influence the similarity score of candidate matches. That is, if the proximity parameter is enabled, non-matching additional terms between matching terms reduces the similarity score of candidate matches.
		PROXIMITY or P - The similarity score influenced by the proximity of query terms in candidate matches.
		$\label{eq:noproximity} \mbox{NOPROXIMITY or } \mbox{N} \mbox{ [DEFAULT] - The similarity score is not influenced by the proximity of query terms in candidate matches.}$



Parameter Name	Default Value	Parameter Description
threshold	20	Starting with Oracle Database 12c Release 2 (12.2), you can provide a threshold value as part of the NDATA operator. Specify a threshold value for percentage of matching grams. The section values containing low percentage of matching grams are ignored. If the threshold value is 20, sections with less than 20% of matching grams are ignored. If this value is lowered, fewer sections are ignored and this leads to a better recall. This threshold value promotes recall over precision as the value is lowered. For example: NDATA (author, LAST First, x, proximity, 10)

Examples for NDATA Operator

An NDATA query on an indexed surname section name that matches terms in the query phrase in any order without influencing the similarity score by the proximity of the jones and smith terms has the form:

```
SELECT entryid, SCORE(1) FROM people WHERE
CONTAINS(idx column, 'NDATA(surname, jones smith)',1)>0;
```

An NDATA query on an indexed surname section name that matches terms in the query phrase in any order and in which similarity scores are influenced by the proximity of the jones and smith terms has the form:

```
SELECT entryid, SCORE(1) FROM people WHERE CONTAINS(idx column, 'NDATA(surname, jones smith,,proximity)',1)>0;
```

An NDATA query on an indexed surname section name that matches terms in the query phrase in-order without influencing the similarity score by the proximity of the jones and smith terms has the form:

```
SELECT entryid, SCORE(1) FROM people WHERE
CONTAINS(idx column, 'NDATA(surname, jones smith, order)',1)>0;
```

An NDATA query on an indexed surname section name that matches terms in the query phrase in-order and in which similarity scores are influenced by the proximity of the jones and smith terms has the form:

```
SELECT entryid, SCORE(1) FROM people WHERE CONTAINS(idx column, 'NDATA(surname, jones smith, order, proximity)',1)>0;
```

Notes

The NDATA query operator does not provide offset information. As such, it cannot be used as a child of WITHIN, NEAR(;), or EQUIV(=), and NDATA sections are ignored by CTX_DOC.HIGHLIGHT, CTX_DOC.SNIPPET, and CTX_DOC.MARKUP.

The NDATA operator is not supported in the CTXCAT grammar. You can use it with other operators, including OR and query templates. You cannot use other query operators inside the NDATA operator.

A use case of the NDATA operator may involve finding a particular entry based on an approximate spelling of a person's full-name and an estimated date-of-birth. Supposing the entries' date-of-births are stored as an SDATA section, user-defined scoring's alternate scoring template can be used to combine the scores of the full-name's NDATA section data and the date-of-birth's SDATA section data.



The name *john smith* is queried for the section specified by the fullname <code>section_name</code>. Altering the <code>NDATA</code> operator's score based on the closeness of the <code>SDATA</code> section's date-of-birth to the date 08-NOV-2012 modifies the ranking of matching documents as follows:

Restrictions

The NDATA query operator does not work with CTX_DOC Package procedures. Attempting to use NDATA with CTX_DOC procedures will return an error stating that this is not supported.

3.18 NEAR (;)

Use the NEAR operator to return a score based on the proximity of two or more query terms.

Oracle Text returns higher scores for terms closer together and lower scores for terms farther apart in a document. If a word or term appears more than once in a NEAR query, then the word must appear more than once in the document in order to match.



The NEAR operator works with only word queries. You cannot use NEAR in ABOUT queries.

Syntax

```
NEAR((word1, word2,..., wordn) [, max span [, order [, minreqd]]])
```

Backward compatibility syntax:

word1; word2

word1-n

Specify the terms in the query separated by commas. The query terms can be single words or phrases and may make use of other query operators (see "NEAR with Other Operators").

max_span

Optionally specify the number of words separating the start and end words of a clump. The default is 100. Oracle Text returns an error if you specify a number greater than 100. A clump is the smallest group of words in which all query terms occur. All clumps begin and end with a query term.



For near queries with two terms, max_span is the maximum distance allowed between the two terms. For example, if the document contains "The cat sat on the dog" then you can find *cat* within 3 words of *dog* by using the following query:

```
'near((dog, cat, 3)'
```

If the document contains "The cat and the rabbit sat on the dog" then you can find *cat*, *dog*, and *rabbit* within 6 words by using the following query:

```
'near((cat, dog, rabbit), 6)'
```



The search term rabbit is still included in the max_span calculation. If you specify a max_span of 5 then you cannot find rabbit. Stopwords are also included in the span calculation.

order

Specify TRUE for Oracle Text to search for terms in the order you specify. The default is FALSE. For example, to search for the words *monday*, *tuesday*, and *wednesday* in that order with a maximum clump size of 20, enter the following query:

```
'near((monday, tuesday, wednesday), 20, TRUE)'
```



To specify order, then you must always specify a number for max span.

Oracle Text might return different scores for the same document when you use identical query expressions that have the order flag set differently. For example, Oracle Text might return different scores for the same document when you enter the following queries:

```
'near((dog, cat), 50, FALSE)'
'near((dog, cat), 50, TRUE)'
```

minreqd

Specify the minimum number of query terms that must be present near each other within a given span, for a document to qualify as a match. You must specify a number greater than 1. If the number of terms that must be near each other for a match is not specified, all terms must match. For example, the following query matches documents that contain clusters of words pertaining to fish:

```
'near((fish, shark, ocean, scales, fishing), 10, FALSE, 3)'
```

Here, only three of the query terms must be within a distance of 10 from each other for a match.

NEAR Scoring

The scoring for the NEAR operator combines frequency of the terms with proximity of terms. For each document that satisfies the query, Oracle Text returns a score between 1 and 100 that is proportional to the number of clumps in the document and inversely proportional to the average size of the clumps. This means many small clumps in a document result in higher scores, because small clumps imply closeness of terms.



The number of terms in a query also affects score. Queries with many terms, such as seven, generally need fewer clumps in a document to score 100 than do queries with few terms, such as two.

A *clump* is the smallest group of words in which all query terms occur. All clumps begin and end with a query term. Define clump size with the max_span parameter, as described in this section.

The size of a clump does not include the query terms themselves. So for the query NEAR(DOG, CAT), 1), dog cat will be a match, and dog ate cat will be a match, but dog sat on cat will not be a match.

NEAR with Other Operators

You can use the NEAR operator with other operators such as and or. Scores are calculated in the regular way.

For example, to find all documents that contain the terms *tiger*, *lion*, and *cheetah* where the terms *lion* and *tiger* are within 10 words of each other, enter the following query:

```
'near((lion, tiger), 10) AND cheetah'
```

The score returned for each document is the lower score of the near operator and the term *cheetah*.

You can also use the equivalence operator to substitute a single term in a near query:

```
'near((stock crash, Japan=Korea), 20)'
```

This query asks for all documents that contain the phrase *stock crash* within twenty words of *Japan* or *Korea*.

The following NEAR syntax is now valid:

```
SELECT * FROM docs WHERE CONTAINS(txt, 'near((aterm1 aterm2 ... atermI
OR bterm1 bterm2 ... btermJ
OR cterm1 cterm2 ... ctermK, dterm))') >0
```

There can be any number of ORS in a given NEAR child, and the OR can appear in any of the NEAR children.

The NEAR within NEAR feature allows users to use nested proximity queries. Starting with Oracle Database 12c Release 2 (12.2), the distance between phrases is measured from the closest words in the phrases. For example, if the document contains the phrases `Lorem ipsum dolor sit amet' and `Sed ut perspiciatis unde omnis', rather than measuring the distance of these two phrases as the distance between `Lorem' and `Sed', the first two words in the phrases, the distance is measured from `amet' and 'Sed'. The distance between phrases is the so-called Hausdorff measure.

```
SELECT * FROM docs
WHERE CONTAINS(txt, 'near((near((term1, term2),5), term3), 100)')>0
```

This query returns documents where term1 and term2 are near within a 5 token window, and the phrase containing term1 and term2 is within a 100 token window from term3. The distance between term3 and the phrase containing term1 and term2 is computed based on the Hausdorff measure.

Mixing the semicolon and NEAR syntax is not supported and throws an error. That is, the queries "near((a;b,c),3)" or "near((a,b));c" will be disallowed.

The following operators also work with NEAR and ; :



- EOUIV
- All expansion operators that produce words, phrases, or EQUIV. These include:
 - soundex
 - fuzzy
 - wildcards
 - stem

Backward Compatibility NEAR Syntax

You can write near queries using the syntax of previous Oracle Text releases. However, in a nested NEAR query, the semicolon operator cannot be used as the inner NEAR. That is, the query $\lceil ((a;d),f),3 \rceil$ produces a syntax error. The semicolon operator can be used as the outermost NEAR in a nested NEAR query.

For example, to find all documents where *lion* occurs near *tiger*, write:

```
'lion near tiger'
```

or with the semi-colon as follows:

'lion; tiger'

This guery is equivalent to the following guery:

```
'near((lion, tiger), 100, FALSE)'
```



Only the syntax of the NEAR operator is backward compatible. In the example, the score returned is calculated using the clump method as described in this section.

Highlighting with the NEAR Operator

When you use highlighting and your query contains the near operator, all occurrences of all terms in the query that satisfy the proximity requirements are highlighted. Highlighted terms can be single words or phrases.

For example, assume a document contains the following text:

Chocolate and vanilla are my favorite ice cream flavors. I like chocolate served in a waffle cone, and vanilla served in a cup with caramel syrup.

If the query is near((chocolate, vanilla)), 100, FALSE), the following is highlighted:

```
<<Chocolate>> and <<vanilla>> are my favorite ice cream flavors. I like <<chocolate>> served in a waffle cone, and <<vanilla>> served in a cup with caramel syrup.
```

However, if the query is near((chocolate, vanilla)), 4, FALSE), only the following is highlighted:

<<Chocolate>> and <<vanilla>> are my favorite ice cream flavors. I like chocolate served in a waffle cone, and vanilla served in a cup with caramel syrup.



See Also:

CTX_DOC Package for more information about the procedures for highlighting

Section Searching and NEAR

Use the NEAR operator with the WITHIN operator for section searching as follows:

```
'near((dog, cat), 10) WITHIN Headings'
```

When evaluating expressions such as these, Oracle Text looks for clumps that lie entirely within the given section.

In this example, only those clumps that contain *dog* and *cat* that lie entirely within the section *Headings* are counted. That is, if the term *dog* lies within *Headings* and the term *cat* lies five words from *dog*, but outside of *Headings*, this pair of words does not satisfy the expression and is not counted.

3.19 NEAR2

Use the NEAR2 operator to perform position—based scoring and length normalization to help improve relevancy.

The NEAR2 operator divides a document into segments based on the given query. Then, it classifies each segment based on the primary features and scores them based on the secondary features. The primary features that are used are as follows:

- Phrase Hits
- Partial Phrase Hits
- Ordered Near Hits
- Unordered Near Hits
- AND Hits

The secondary features are as follows:

- Excess Span
- Start Position
- Longest Partial Phrase

Syntax

```
NEAR2((word1, word2,...,wordn),max_span, phrase_weight,
partial_phrase_weight, ordered_near_weight, unordered_near_weight)
```

All or none of the weights must be provided. When the weights are provided, the NEAR2 operator works in the weighted-average mode. The weights are integers between 0 and 10.

word1-n

Specify the terms in the query separated by commas. The query terms can be single words or phrases and can use other query operators (see "NEAR with Other Operators"). Only the word list is mandatory.



max_span

Optionally, specify the size of the biggest clump. The default is 50. Oracle Text returns an error if you specify a number greater than 50.

A clump is the smallest group of words in which all query terms occur. All clumps begin and end with a query term.

For near queries with two terms, max_span is the maximum distance allowed between the two terms. For example, to query on *dog* and *cat* where *dog* is within 6 words of *cat*, enter the following query:

'near((dog, cat), 6)'

phrase weight

Determine the weight of the phrase primary feature when in weighted-average mode. This is a qualitative weight, which is mapped to an internal weight.

partial_phrase_weight

Determine the weight of the partial phrase primary feature when in weighted-average mode. This is a qualitative weight.

ordered_near_weight

Determine the weight of the ordered near primary feature when in weighted-average mode. This is a qualitative weight.

unordered_near_weight

Determine the weight of the unordered near primary feature when in weighted-average mode. This is a qualitative weight.

and weight

Determine the weight of the AND primary feature when in weighted average mode. This is a qualitative weight.

Related Topics

Oracle Text Application Developer's Guide

3.20 NOT (~)

Use the NOT operator to search for documents that contain one guery term and not another.

Syntax

Syntax	Description
term1~term2	Returns documents that contain term1 and not term2.
term1 not term2	

Examples for NOT Operator

To obtain the documents that contain the term *animals* but not *dogs*, use the following expression:

'animals ~ dogs'

Similarly, to obtain the documents that contain the term *transportation* but not *automobiles* or *trains*, use the following expression:

'transportation not (automobiles or trains)'





The NOT operator does not affect the scoring produced by the other logical operators.

Related Topics

"MINUS (-)"

3.21 OR (|)

Use the OR operator to search for documents that contain at least one occurrence of any of the query terms. The OR operator returns documents that contain any of the query terms, while the AND operator returns documents that contain all query terms.

Syntax

Syntax	Description
term1 term2 term1 or term2	Returns documents that contain <i>term1</i> or <i>term2</i> . Returns the maximum score of its operands. At least one term must exist; higher score taken.

Examples for OR Operator

To obtain the documents that contain the term *cats* or the term *dogs*, use either of the following expressions:

```
'cats | dogs'
'cats OR dogs'
```

Scoring

In an OR query, the score returned is the score for the highest query term. In the example, if the scores for *cats* and *dogs* is 30 and 40 within a document, the document scores 40.

Related Topics

"AND (&)"

3.22 Preferred Term (PT)

Use the preferred term operator (PT) to replace a term in a query with the preferred term that has been defined in a thesaurus for the term.

Syntax

Syntax	Description
PT(term[,thes])	Replaces the specified word in a query with the preferred term for term.



term

Specify the operand for the preferred term operator. term is replaced by the preferred term defined for the term in the specified thesaurus. However, if no PT entries are defined for the term, term is not replaced in the query expression and term is the result of the expansion. You cannot specify expansion operators in the term argument.

thes

Specify the name of the thesaurus used to return the expansions for the specified term. The thes argument is optional and has a default value of DEFAULT. As a result, a thesaurus named DEFAULT *must* exist in the thesaurus tables before using any of the thesaurus operators.

Example for PT Operator

The term *automobile* has a preferred term of *car* in a thesaurus. A PT query for *automobile* returns all documents that contain the word *car*. Documents that contain the word *automobile* are not returned.

Related Topics

CTX_THES.PT in CTX_THES Package form more information on browsing the preferred terms in your thesaurus

3.23 Related Term (RT)

Use the related term operator (RT) to expand a query to include all related terms that have been defined in a thesaurus for the term.

Syntax

Syntax	Description
RT(term[,thes])	Expands a query to include all the terms defined in the thesaurus as a related term for term.

term

Specify the operand for the related term operator. term is expanded to include term and all the related entries defined for term in thes.

You cannot specify expansion operators in the term argument.

thes

Specify the name of the thesaurus used to return the expansions for the specified term. The thes argument is optional and has a default value of DEFAULT. As a result, a thesaurus named DEFAULT *must* exist in the thesaurus tables before using any of the thesaurus operators.

Example for RT Operator

The term *dog* has a related term of *wolf*. An RT query for *dog* returns all documents that contain the word *dog* and *wolf*.

Related Topics

CTX_THES.RT in CTX_THES Package for more information on browsing the related terms in your thesaurus



3.24 SDATA

Use the SDATA operator to perform tests on SDATA sections and columns, which contain structured data values.

SDATA sections speed up mixed querying and ordering. This operator provides structured predicate support for CONTAINS, which extends non-SQL interfaces such as count_hits or the result set interface.

SDATA operators should only be used as descendants of AND operators that also have non-SDATA children.

SDATA queries perform on string or numeric literals, and on date strings. The string literal and date string are enclosed within single or double quote characters. The numeric value is not enclosed in quote characters, and must conform to the SQL format of NUMBER. For example:

```
CONTAINS(text, "dog and SDATA(category = ''news'')")>0 ...

SDATA(rating between 1.2 and 3.4) ...

SDATA(author LIKE 'FFORDE%') ...

SDATA(date >='2005-09-18') ...
```

Closed parentheses are permitted, as long as they are enclosed in single or double quotes.

The SDATA operator can be used in query templates.

Syntax

Syntax	Operators
SData	:= "SDATA" "(" SDataPredicate ")"
SDataPredicate	:= section_name SDataTest
SDataTest	:= <sdatasingleop sdataliteral=""> SDataBetweenOp <"is" ("not")? "null"></sdatasingleop>
SDataSingleOp	:= ("<" "<=" "=" ">=" ">" "!=" "<>" "like")
SDataBetweenOp	:= "between" SDataLiteral "and" SDataLiteral
SDataLiteral	:= numeric_literal """ string_literal """ """ date_string """

section name

The name of the SDATA section(s) on which to search and perform the test, or check.

SDataLiteral

The value of the SDATA section. This must be either a string literal, numeric literal, or a date string.

The SDATA operator returns a score of 100 if the enclosed predicate returns TRUE, and returns 0 otherwise. In the case of a NULL value, the SDATA operator returns a score of 0 (since in SQL it would not return TRUE).

Multi-valued semantics are not defined, as multi-valued SDATA sections are not supported. Comparison of strings is case sensitive. The BINARY collation is always used.



Note:

For the SDATA operator on SDATA sections that are mapped to a DATE FILTER BY column, the SDATA value must follow the Date format: YYYY-MM-DD or YYYY-MM-DD HH24:MI:SS. Otherwise, the expected rows will not be returned. If the time component is omitted, it will default to 00:00:00, according to SQL semantics. This Date format is always used, regardless of the setting of the NLS_DATE_FORMAT environment variable.

Example for SDATA Operator

Suppose that you want to query for books in the fiction category that contain the word *summer*. Assuming that an SDATA section called CATEGORY has been declared, you can query as follows:

```
SELECT id FROM idx_docs
  WHERE CONTAINS(text, 'summer AND SDATA(category = "fiction")')>0
```

Restrictions

- An error is raised if the section name is not a defined SDATA section. The source of the section (for example, tag versus column) is not important.
- The syntax precludes RHS SDATA and expressions.
- SDATA operators cannot be children of WITHIN, INPATH, HASPATH, or NEAR.
- The data type of the named SDATA section must be compatible with the literal provided (and the operator, for example, LIKE) or an error is raised.
- SDATA operators are not supported in CTXRULE query documents.
- SDATA operators have no effect on highlighting.

Notes

Stoplists do not affect string-value SDATA sections, that is, if a stopword is present within an SDATA section, then the token will still be indexed and can be gueried using the SDATA operator.

Oracle recommends using SDATA operators only as descendants of AND operators that also have non-SDATA children. Essentially, use SDATA operators as secondary (that is, checking or non-driving) criteria. For instance, "find documents with DOG that also have price > 5", rather than "find documents with rating > 4". Other usage may operate properly, but may not have optimal performance.

The following examples are consistent with recommended use:

```
dog \& SDATA(foo = 5)
```

The SDATA is a child of an AND operator that also has non-SDATA children.

```
dog \& (SDATA(foo = 5) | SDATA(x = 1))
```

Although the SDATA operators here are children of OR, they are still descendants of an AND operator with non-SDATA children.

The following examples show use that is not recommended:

```
SDATA (foo = 5)
```

Here, SDATA is the only criteria and, therefore, the driving criteria.



```
dog \mid SDATA(bar = 9)
```

The SDATA in this example is a child of an OR operator rather than an AND.

```
SDATA(foo = 5) & SDATA(bar = 7)
```

While both SDATA operators in this example are descendants of AND, this AND operator does not have non-SDATA children.

Related Topics

ADD SDATA COLUMN

Use this procedure to map the FILTER BY or ORDER BY column (named in *column_name*) to the SDATA section (named in *section_name*). By default, all FILTER BY columns are mapped as SDATA.

ADD SDATA SECTION

This procedure adds an SDATA section to a section group. By default, all FILTER BY columns are mapped as SDATA.

UPDATE SDATA

UPDATE_SDATA is an index API that modifies the specified SDATA values in the index. This API does not store or modify column values in a base table, where the base table column may have been used as an SDATA section.

- CTX SECTIONS
- Oracle Text Views

This is a list of all the views provided by Oracle Text.

Oracle Text Application Developer's Guide

3.25 soundex (!)

Use the soundex (!) operator to expand queries to include words that have similar sounds; that is, words that sound like other words.

This function enables comparison of words that are spelled differently, but sound alike in English. The SOUNDEX operator algorithm uses heuristic methods, so results may vary based on your query words.

Syntax

Syntax	Description	
!term	Expands a query to include all terms that sound the same as the specified term (English-language text only).	

Example for Soundex (!) Operator



Language

Soundex works best for languages that use a 7-bit character set, such as English. It can be used, with lesser effectiveness, for languages that use an 8-bit character set, such as many Western European languages.

If you have base-letter conversion specified for a text column and the query expression contains a soundex operator, then Oracle Text operates on the base-letter form of the query.

3.26 stem (\$)

Use the stem (\$) operator to search for terms that have the same linguistic root as the query term.

If you use <code>BASIC_LEXER</code> to index your language, then you can improve stemming performance by using the <code>index</code> stems attribute of <code>BASIC_LEXER</code>.

Japanese stemming is supported with JAPANESE LEXER.

Specify your stemming language with the BASIC WORDLIST wordlist preference.

Syntax

Syntax	Description	
\$term	Expands a query to include all terms having the same stem or root word as the specified term.	

Examples for Stem (\$) Operator

Input	Expands To
\$scream	scream screaming screamed
\$distinguish	distinguish distinguished distinguishes
\$guitars	guitars guitar
\$commit	commit committed
\$cat	cat cats
\$sing	sang sung sing

Behavior with Stopwords

If stem returns a word designated as a stopword, the stopword is not included in the query or highlighted by CTX QUERY.HIGHLIGHT or CTX QUERY.MARKUP.

Related Topics

BASIC_LEXER Attributes

These are the attributes supported for the BASIC LEXER component.

BASIC_WORDLIST

Use ${\tt BASIC_WORDLIST}$ to enable stemming and fuzzy matching or to create prefix indexes with Text indexes.



JAPANESE LEXER

Identifies tokens in Japanese for creating Oracle Text indexes. Offers advantages over <code>JAPANESE_VGRAM_LEXER</code>, such as generates a smaller index, has a better query response time, and generates real word tokens resulting in better query precision.

Oracle Text Indexing Elements

3.27 Stored Query Expression (SQE)

Use the SQE operator to call a stored query expression created with the \mathtt{CTX} QUERY. STORE_SQE procedure.

Stored query expressions can be used for creating predefined bins for organizing and categorizing documents or to perform iterative queries, in which an initial query is refined using one or more additional queries.

Syntax

Syntax	Description
SQE(SQE_name)	Returns the results for the stored query expression SQE_name.

Examples for SQE Operator

To create an SQE named disasters, use CTX QUERY.STORE_SQE as follows:

```
begin
ctx_query.store_sqe('disasters', 'hurricane or earthquake or blizzard');
end;
```

This stored query expression returns all documents that contain either *hurricane*, *earthquake* or *blizzard*.

This SQE can then be called within a query expression as follows:

```
SELECT SCORE(1), docid FROM news
WHERE CONTAINS(resume, 'sqe(disasters)', 1) > 0
ORDER BY SCORE(1);
```

Limitations

Up to 100 stored query expressions (SQEs) can be stored in a single Text query. If a Text query has more than 100 SQEs, including nested SQEs, then the query fails and error DRG-50949 is raised.

Related Topic

"STORE_SQE"

3.28 SYNonym (SYN)

Use the synonym operator (SYN) to expand a query to include all the terms that have been defined in a thesaurus as synonyms for the specified term.

Syntax

Syntax	Description	
SYN(term[,thes])	Expands a query to include all the terms defined in the thesaurus as synonyms for term.	

term

Specify the operand for the synonym operator. term is expanded to include term and all the synonyms defined for term in thes.

You cannot specify expansion operators in the term argument.

thes

Specify the name of the thesaurus used to return the expansions for the specified term. The thes argument is optional and has a default value of <code>DEFAULT</code>. A thesaurus named <code>DEFAULT</code> must exist in the thesaurus tables if you use this default value.

Examples for SYN Operator

The following query expression returns all documents that contain the term *dog* or any of the synonyms defined for *dog* in the DEFAULT thesaurus:

'SYN (dog) '

Compound Phrases in Synonym Operator

Expansion of compound phrases for a term in a synonym query are returned as AND conjunctives.

For example, the compound phrase *temperature* + *measurement* + *instruments* is defined in a thesaurus as a synonym for the term *thermometer*. In a synonym query for *thermometer*, the query is expanded to:

{thermometer} OR ({temperature}&{measurement}&{instruments})

Related Topics

CTX_THES.SYN in CTX_THES Package for more information on browsing the synonym terms in your thesaurus

3.29 threshold (>)

Use the threshold operator (>) in two ways:

- at the expression level
- at the query term level

The threshold operator at the expression level eliminates documents in the result set that score below a threshold number.

The threshold operator at the query term level selects a document based on how a term scores in the document.



Syntax

Syntax	Description
expression>n term>n	Returns only those documents in the result set that score above the threshold <i>n</i> .
	Within an expression, returns documents that contain the query term with score of at least <i>n</i> .

Examples for Threshold (>) Operator

At the expression level, to search for documents that contain *relational databases* and to return only documents that score greater than 75, use the following expression:

At the query term level, to select documents that have at least a score of 30 for *lion* and contain *tiger*, use the following expression:

3.30 Translation Term (TR)

Use the translation term operator (TR) to expand a query to include all defined foreign language equivalent terms.

Syntax

Syntax	Description	
TR(term[, lang, [thes]])	Expands <i>term</i> to include all the foreign equivalents that are defined for <i>term</i> .	

term

Specify the operand for the translation term operator. term is expanded to include all the foreign language entries defined for term in thes. You cannot specify expansion operators in the term argument.

lang

Optionally, specify which foreign language equivalents to return in the expansion. The language you specify must match the language as defined in thes. (You may specify only one language at a time.) If you omit this parameter or specify it as ALL, the system expands to use all defined foreign language terms.

thes

Optionally, specify the name of the thesaurus used to return the expansions for the specified term. The thes argument has a default value of DEFAULT. As a result, a thesaurus named DEFAULT *must* exist in the thesaurus tables before you can use any of the thesaurus operators.



^{&#}x27;relational databases > 75'

^{&#}x27;(lion > 30) and tiger'



If you specify thes, then you must also specify lang.

Examples for TR Operator

Consider a thesaurus MY THES with the following entries for cat:

```
cat
SPANISH: gato
FRENCH: chat
```

To search for all documents that contain *cat* and the spanish translation of *cat*, enter the following query:

```
'tr(cat, spanish, my_thes)'
```

This guery expands to:

```
'{cat}|{gato}'
```

Related Topics

CTX_THES.TR in CTX_THES Package for more information on browsing the related terms in your thesaurus

3.31 Translation Term Synonym (TRSYN)

Use the translation term synonym operator (TRSYN) to expand a query to include all the defined foreign equivalents of the query term, the synonyms of query term, and the foreign equivalents of the synonyms.

Syntax

Syntax	Description	
TRSYN(term[, lang, [thes]])	Expands term to include foreign equivalents of term, the synonyms of term, and the foreign equivalents of the synonyms.	

term

Specify the operand for this operator. term is expanded to include all the foreign language entries and synonyms defined for term in thes. You cannot specify expansion operators in the term argument.

lang

Optionally, specify which foreign language equivalents to return in the expansion. The language you specify must match the language as defined in thes. If you omit this parameter, the system expands to use all defined foreign language terms.

thes

Optionally, specify the name of the thesaurus used to return the expansions for the specified term. The thes argument has a default value of DEFAULT. As a result, a thesaurus named DEFAULT *must* exist in the thesaurus tables before you can use any of the thesaurus operators.





If you specify thes, then you must also specify lang.

Examples for TRSYN Operator

Consider a thesaurus MY THES with the following entries for cat:

```
cat
SPANISH: gato
FRENCH: chat
SYN lion
SPANISH: leon
```

To search for all documents that contain *cat*, the spanish equivalent of *cat*, the synonym of *cat*, and the spanish equivalent of *lion*, enter the following query:

```
'trsyn(cat, spanish, my thes)'
```

This query expands to:

```
'{cat}|{gato}|{lion}|{leon}'
```

Related Topics

CTX_THES.TRSYN in CTX_THES Package for more information on browsing the translation and synonym terms in your thesaurus

3.32 Top Term (TT)

Use the top term operator (TT) to replace a term in a query with the *top term* that has been defined for the term in the standard hierarchy (Broader Term [BT], Narrower Term [NT]) in a thesaurus. A top term is the broadest conceptual term related to a given query term. For example, a thesaurus might define the following hierarchy:

```
DOG
BT1 CANINE
BT2 MAMMAL
BT3 VERTEBRATE
BT4 ANIMAL
```

The top term for *dog* in this thesaurus is *animal*.

Top terms in the generic (BTG, NTG), partitive (BTP, NTP), and instance (BTI, NTI) hierarchies are not returned.

Syntax

Syntax	Description	
TT(term[,thes])	Replaces the specified word in a query with the top term in the standard hierarchy (BT, NT) for term.	



term

Specify the operand for the top term operator. term is replaced by the top term defined for the term in the specified thesaurus. However, if no TT entries are defined for term, term is not replaced in the query expression and term is the result of the expansion. You cannot specify expansion operators in the term argument.

thes

Specify the name of the thesaurus used to return the expansions for the specified term. The thes argument is optional and has a default value of <code>DEFAULT</code>. A thesaurus named <code>DEFAULT</code> must exist in the thesaurus tables if you use this default value.

Example for TT Operator

The term *dog* has a top term of *animal* in the standard hierarchy of a thesaurus. A TT query for *dog* returns all documents that contain the phrase *animal*. Documents that contain the word *dog* are not returned.

Related Topics

CTX_THES.TT for more information on browsing the top terms in your thesaurus

3.33 weight (*)

The weight operator multiplies the score by the given factor, topping out at 100 when the score exceeds 100. For example, the query *cat*, *dog*2* sums the score of *cat* with twice the score of *dog*, topping out at 100 when the score is greater than 100.

In expressions that contain more than one query term, use the weight operator to adjust the relative scoring of the query terms. Reduce the score of a query term by using the weight operator with a number less than 1; increase the score of a query term by using the weight operator with a number greater than 1 and less than 10.

The weight operator is useful in ACCUMulate (__), AND (&), or OR (|) queries when the expression has more than one query term. With no weighting on individual terms, the score cannot tell which of the query terms occurs the most. With term weighting, you can alter the scores of individual terms and hence make the overall document ranking reflect the terms you are interested in.

Syntax

Syntax	Description
term*n	Returns documents that contain term. Calculates score by multiplying the raw score of term by n , where n is a number from 0.1 to 10.

Examples for Weight (*) Operator

Suppose you have a collection of sports articles. You are interested in the articles about Brazilian soccer. It turns out that a regular query on *soccer or Brazil* returns many high ranking articles on US soccer. To raise the ranking of the articles on Brazilian soccer, enter the following query:

'soccer or Brazil*3'

Table 3-3 illustrates how the weight operator can change the ranking of three hypothetical documents A, B, and C, which all contain information about soccer. The columns in the table show the total score of four different query expressions on the three documents.



Table 3-3 Score Samples

Document	soccer	Brazil	soccer or Brazil	soccer or Brazil*3
A	20	10	20	30
В	10	30	30	90
С	50	20	50	60

The score in the third column containing the query *soccer or Brazil* is the score of the highest scoring term. The score in the fourth column containing the query *soccer or Brazil*3* is the larger of the score of the first column *soccer* and of the score *Brazil* multiplied by three, *Brazil*3*.

With the initial query of *soccer or Brazil*, the documents are ranked in the order C B A. With the query of *soccer or Brazil*3*, the documents are ranked B C A, which is the preferred ranking.

Weights can be added to multiple terms. The query *Brazil OR* (soccer AND *Brazil*)*3 will increase the relative scores for documents that contain both soccer and *Brazil*.

3.34 wildcards (% _)

Wildcard characters can be used in query expressions to expand word searches into pattern searches. When a wildcard is used on its own, for example, "DOG %" or ".%" or "%" by itself, it is treated as a stopword.

The wildcard characters are as follows:

Wildcard Character	Description
%	The percent wildcard can appear any number of times at any part of the search term. The search term will be expanded into an equivalence list of terms. The list consists of all terms in the index that match the wildcarded term, with zero or more characters in place of the percent character.
-	The underscore wildcard specifies a single position in which any character can occur.

The total number of wildcard expansions from all words in a query containing unescaped wildcard characters cannot exceed the maximum number of expansions specified by the BASIC_WORDLIST attribute WILDCARD_MAXTERMS. For more information, see "BASIC_WORDLIST".

Note:

- When a wildcard is used on its own, it is treated as a stopword.
- When a wildcard expression translates to a stopword, the stopword is not included in the query and not highlighted by CTX_DOC.HIGHLIGHT or CTX_DOC.MARKUP.

Right-Truncated Queries

Right truncation involves placing the wildcard on the right-hand-side of the search string.

For example, the following query expression finds all terms beginning with the pattern scal:

'scal%'

Left- and Double-Truncated Queries

Left truncation involves placing the wildcard on the left-hand-side of the search string.

To find words such as king, wing or sing, write the query as follows:

'_ing'

For all words that end with ing, enter:

'%ing'

Combine left-truncated and right-truncated searches to create double-truncated searches. The following query finds all documents that contain words that contain the substring *%benz%*

'%benz%'

Improving Wildcard Query Performance

Improve wildcard query performance by adding a substring or prefix index.

When your wildcard queries are left- and double-truncated, you can improve query performance by creating a substring index. Substring indexes improve query performance for all types of left-truncated wildcard searches such as *%ed*, _ing, or *%benz%*.

When your wildcard queries are right-truncated, you can improve performance by creating a prefix index. A prefix index improves query performance for wildcard searches such as *to*%.

See Also:

"BASIC_WORDLIST" in Oracle Text Indexing Elements for more information about creating substring and prefix indexes

3.35 WITHIN

Use the WITHIN operator to narrow a query down into document sections. Document sections can be one of the following:

- Zone sections
- Field sections
- Attribute sections
- Special sections (sentence or paragraph)



Syntax

Syntax	Description
expression WITHIN section	Searches for expression within the predefined zone, field, or attribute section.
	If section is a zone, expression can contain one or more WITHIN operators (nested WITHIN) whose section is a zone or special section.
	If section is a field or attribute section, expression cannot contain another WITHIN operator.
expression WITHIN SENTENCE	Searches for documents that contain expression within a sentence. Specify an AND or NOT query for expression.
	The expression can contain one or more WITHIN operators (nested WITHIN) whose section is a zone or special section.
expression WITHIN PARAGRAPH	Searches for documents that contain expression within a paragraph. Specify an AND or NOT query for expression.
	The expression can contain one or more WITHIN operators (nested WITHIN) whose section is a zone or special section.

WITHIN Limitations

The WITHIN operator has the following limitations:

- You cannot embed the WITHIN clause in a phrase. For example, you cannot write: term1
 WITHIN section term2
- Because WITHIN is a reserved word, you must escape the word with braces to search on it.

WITHIN Operator Examples

Querying Within Zone Sections

To find all the documents that contain the term *San Francisco* within the section *Headings*, write the query as follows:

To find all the documents that contain the term *sailing* and contain the term *San Francisco* within the section *Headings*, write the query in one of two ways:

Compound Expressions with WITHIN

To find all documents that contain the terms *dog* and *cat* within the same section *Headings*, write the query as follows:

This query is logically different from:



^{&#}x27;San Francisco WITHIN Headings'

^{&#}x27;(San Francisco WITHIN Headings) and sailing'

^{&#}x27;sailing and San Francisco WITHIN Headings'

^{&#}x27;(dog and cat) WITHIN Headings'

^{&#}x27;dog WITHIN Headings and cat WITHIN Headings'

This query finds all documents that contain *dog* and *cat* where the terms *dog* and *cat* are in *Headings* sections, regardless of whether they occur in the same *Headings* section or different sections.

Near with WITHIN

To find all documents in which *dog* is near *cat* within the section *Headings*, write the query as follows:

'dog near cat WITHIN Headings'



The near operator has higher precedence than the WITHIN operator so braces are not necessary in this example. This query is equivalent to (dog near cat) WITHIN Headings.

Nested WITHIN Queries

You can nest the within operator to search zone sections within zone sections.

For example, assume that a document set had the zone section AUTHOR nested within the zone BOOK section. Write a nested WITHIN query to find all occurrences of scott within the AUTHOR section of the BOOK section as follows:

'(scott WITHIN AUTHOR) WITHIN BOOK'

Querying Within Field Sections

The syntax for querying within a field section is the same as querying within a zone section. The syntax for most of the examples given in the previous section, "Querying Within Zone Sections", apply to field sections.

However, field sections behave differently from zone sections in terms of

- Visibility: Make text within a field section invisible.
- Repeatability: WITHIN queries cannot distinguish repeated field sections.
- Nestability: You cannot enter a nested WITHIN query with a field section.

The following sections describe these differences.

Visible Flag in Field Sections

When a field section is created with the visible flag set to FALSE in CTX_DDL.ADD_FIELD_SECTION, the text within a field section can only be queried using the WITHIN operator.

For example, assume that <code>TITLE</code> is a field section defined with visible flag set to <code>FALSE</code>. Then the query dog without the <code>WITHIN</code> operator will not find a document containing:

<TITLE>The dog</TITLE> I like my pet.

To find such a document, use the WITHIN operator as follows:

'dog WITHIN TITLE'



Alternatively, set the visible flag to TRUE when you define TITLE as a field section with CTX DDL.ADD FIELD SECTION.



"ADD_FIELD_SECTION" in CTX_DDL Package for more information about creating field sections

Repeated Field Sections

WITHIN queries cannot distinguish repeated field sections in a document. For example, consider the document with the repeated section <author>:

```
<author> Charles Dickens </author> <author> Martin Luther King </author>
```

Assuming that <author> is defined as a field section, a query such as (*charles and martin*) within author returns the document, even though these words occur in separate tags.

To have WITHIN queries distinguish repeated sections, define the sections as zone sections.

Nested Field Sections

You cannot enter a nested WITHIN query with field sections. Doing so raises an error.

Querying Within Sentence or Paragraphs

Querying within sentence or paragraph boundaries is useful to find combinations of words that occur in the same sentence or paragraph. To query sentence or paragraphs, you must first add the special section to your section group before you index. Do so with CTX DDL.ADD SPECIAL SECTION.

To find documents that contain *dog* and *cat* within the same sentence:

```
'(dog and cat) WITHIN SENTENCE'
```

To find documents that contain *dog* and *cat* within the same paragraph:

```
'(dog and cat) WITHIN PARAGRAPH'
```

To find documents that contain sentences with the word *dog* but not *cat*:

```
'(dog not cat) WITHIN SENTENCE'
```

Querying Within Attribute Sections

Query within attribute sections when you index with either ${\tt XML_SECTION_GROUP}$ or ${\tt AUTO_SECTION_GROUP}$ as your section group type.

Assume you have an XML document as follows:

```
<book title="Tale of Two Cities">It was the best of times.
```

Define the section title@book to be the attribute section title. Do so with the CTX_DLL.ADD_ATTR_SECTION procedure or dynamically after indexing with ALTER INDEX.



Note:

When you use the AUTO_SECTION_GROUP to index XML documents, the system automatically creates attribute sections and names them in the form attribute@tag.

If you use the ${\tt XML_SECTION_GROUP},$ you can name attribute sections anything with CTX DDL.ADD ATTR SECTION.

To search on *Tale* within the attribute section title, enter the following query:

'Tale WITHIN title'

Constraints for Querying Attribute Sections

The following constraints apply to querying within attribute sections:

 Regular queries on attribute text do not hit the document unless qualified in a within clause. Assume you have an XML document as follows:

```
<book title="Tale of Two Cities">It was the best of times.
```

A query on *Tale* by itself does not produce a hit on the document unless qualified with WITHIN title@book. (This behavior is like field sections when you set the visible flag set to false.)

- You cannot use attribute sections in a nested WITHIN query.
- Phrases ignore attribute text. For example, if the original document looked like:

Now is the time for all good <word type="noun"> men </word> to come to the aid.

Then this document would hit on the regular query *good men*, ignoring the intervening attribute text.

• WITHIN queries can distinguish repeated attribute sections. This behavior is like zone sections but unlike field sections. For example, you have a document as follows:

```
<book title="Tale of Two Cities">It was the best of times.<br/><book title="Of Human Bondage">The sky broke dull and gray.
```

Assume that book is a zone section and book@author is an attribute section. Consider the query:

'(Tale and Bondage) WITHIN book@author'

This query does *not* hit the document, because *tale* and *bondage* are in different occurrences of the attribute section book@author.

Notes

Section Names

The WITHIN operator requires you to know the name of the section you search. A list of defined sections can be obtained using the CTX_SECTIONS or CTX_USER_SECTIONS views.

Section Boundaries

For special and zone sections, the terms of the query must be fully enclosed in a particular occurrence of the section for the document to satisfy the query. This is not a requirement for field sections.



For example, consider the query where *bold* is a zone section:

```
'(dog and cat) WITHIN bold'
```

This query finds:

dog cat

but it does not find:

dogcat

This is because dog and cat must be in the same *bold* section.

This behavior is especially useful for special sections, where

```
'(dog and cat) WITHIN sentence'
```

means find *dog* and *cat* within the same sentence.

Field sections on the other hand are meant for non-repeating, embedded metadata such as a title section. Queries within field sections cannot distinguish between occurrences. All occurrences of a field section are considered to be parts of a single section. For example, the query:

```
(dog and cat) WITHIN title
```

can find a document like this:

<TITLE>dog</TITLE><TITLE>cat</TITLE>

In return for this field section limitation and for the overlap and nesting limitations, field section queries are generally faster than zone section queries, especially if the section occurs in every document, or if the search term is common.

3.36 Supported Oracle Text CONTAINS Query Operators for In-Memory Full Text Search

You can query for simple words and phrases using the CONTAINS operator when In-Memory full text search is enabled.

For querying a text column, only the following Oracle Text query operators are supported:

- AND
- OR
- NOT
- NEAR

For querying a JSON column, the following Oracle Text query operators are also supported:

- HASPATH
- INPATH





Oracle Text Application Developer's Guide for more information about In-Memory full text search and JSON full text search



4

Special Characters in Oracle Text Queries

This chapter describes the special characters that can be used in Text queries. In addition, it provides a list of the words and characters that Oracle Text treats as reserved words and characters.

The following topics are covered in this chapter:

- Grouping Characters
- Escape Characters
- Reserved Words and Characters

4.1 Grouping Characters

The grouping characters control operator precedence by grouping query terms and operators in a query expression. The grouping characters are described in Table 4-1.

Table 4-1 Characters for Grouping Query Terms

Grouping Character	Description
()	The parentheses characters serve to group terms and operators found between the characters
[]	The bracket characters serve to group terms and operators found between the characters; however, they prevent penetrations for the expansion operators (fuzzy, soundex, stem).

The beginning of a group of terms and operators is indicated by an open character from one of the sets of grouping characters. The ending of a group is indicated by the occurrence of the appropriate close character for the open character that started the group. Between the two characters, other groups may occur.

For example, the open parenthesis indicates the beginning of a group. The first close parenthesis encountered is the end of the group. Any open parentheses encountered before the close parenthesis indicate nested groups.

4.2 Escape Characters

To query on words or symbols that have special meaning in query expressions such as *and & or*| *accum*, you must escape them. There are two ways to escape characters in a query expression, as described in Table 4-2.

Table 4-2 Characters for Escaping Query Terms

Escape Character	Description
0	Use braces to escape a string of characters or symbols. Everything within a set of braces in considered part of the escape sequence.
	When you use braces to escape a single character, the escaped character becomes a separate token in the query.
\	Use the backslash character to escape a single character or symbol. Only the character immediately following the backslash is escaped. For example, a query of <i>bluel-green</i> matches <i>blue-green</i> and <i>blue green</i> .

In the following examples, an escape sequence is necessary because each expression contains a Text operator or reserved symbol:

```
'high\-voltage'
'{high-voltage}'
'XY\&Z'
'{XY&Z}'
```

In the first example, the query matches high-voltage or high voltage.

Note that in the second example, a query on XY&Z will return 'XY Z', 'XY-Z', 'XY*Z', and so forth, as well as 'XY&Z'. This is because non-alphabetic characters are treated as whitespace (so XY&Z is treated as 'XY Z'). To match only XY&Z, you must declare & as a printjoin. (If you do, however, XY&Z will not match 'XY & Z'.) For more on printjoins, see BASIC LEXER.



If you use braces to escape an individual character within a word, the character is escaped, but the word is broken into three tokens. For example, a query written as <code>high{-}voltage</code> searches for <code>high - voltage</code>, with the space on either side of the hyphen.

Querying Escape Characters

The open brace { signals the beginning of the escape sequence, and the closed brace } indicates the end of the sequence. Everything between the opening brace and the closing brace is part of the escaped query expression (including any open brace characters). To include the close brace character in an escaped query expression, use }}. To escape the backslash escape character, use \\.

4.3 Reserved Words and Characters

Table 4-3 lists the Oracle Text reserved words and characters that must be escaped when you want to search them in CONTAINS queries. Refer to Table 4-2 for the rule for when to use braces {} or the backslash \ for the escape sequence.

Table 4-3 Reserved Words and Characters

Reserved Words	Reserved Characters	Operator
ABOUT	(none)	ABOUT
ACCUM	,	Accumulate
AND	, &	And
ВТ	(none)	Broader Term
BTG	(none)	Broader Term Generic
BTI	(none)	Broader Term Instance
ВТР	(none)	Broader Term Partitive
EQUIV	=	Equivalence
FUZZY	?	fuzzy
(none)	{}	escape characters (multiple)
(none)	\	escape character (single)
(none)	()	grouping characters
(none)	[]	grouping characters
HASPATH	(none)	HASPATH
INPATH	(none)	INPATH
MDATA	(none)	MDATA
MINUS	-	MINUS
NEAR	•	NEAR
NOT	~	NOT
NT	(none)	Narrower Term
NTG	(none)	Narrower Term Generic
NTI	(none)	Narrower Term Instance
NTP	(none)	Narrower Term Partitive
OR	1	OR
PATTERN	(none)	PATTERN
PT	(none)	Preferred Term
RT	(none)	Related Term
(none)	\$	stem
(none)	!	soundex
SQE	(none)	Stored Query Expression
SYN	(none)	Synonym
(none)	>	threshold
TR	(none)	Translation Term
TRSYN	(none)	Translation Term Synonym
TT	(none)	Top Term
(none)	*	weight
(none)	%	wildcard character (multiple)
(none)	_	wildcard character (single)
WITHIN	(none)	WITHIN



CTX_ADM Package

This chapter contains the following topics.

- About CTX ADM Package Procedures
- MARK FAILED
- RECOVER
- RESET_AUTO_OPTIMIZE_STATUS
- SET PARAMETER

5.1 About CTX_ADM Package Procedures

The CTX_ADM PL/SQL package provides administrative procedures for managing index preferences.

The CTX ADM package contains the following stored procedures.

Name	Description
MARK_FAILED	Changes an index's status from LOADING to FAILED.
RECOVER	Cleans up database objects for deleted Text tables.
RESET_AUTO_OPTIMIZE_STATUS	Resets the CTX_AUTO_OPTIMIZE_STATUS view.
SET_PARAMETER	Sets system-level defaults for index creation.



Only the CTXSYS user can use the procedures in the CTX ADM package.

The APIs in the $\mathtt{CTX}_\mathtt{ADM}$ package do not support identifiers that are prefixed with the schema or the owner name.

5.2 MARK FAILED

Use the MARK FAILED procedure to change the status of an index from LOADING to FAILED.

Under rare circumstances, if CREATE INDEX or ALTER INDEX fails, an index may be left with the status LOADING. When an index is in LOADING status, any attempt to recover using RESUME INDEX is blocked. For this situation, use CTX_ADM.MARK_FAILED to forcibly change the status from LOADING to FAILED so that you can recover the index with RESUME INDEX.

Note:

CTX ADM. MARK FAILED will mark the index or index partition as FAILED, even if they are not marked as LOADING.

You must log on as CTXSYS to run CTX ADM. MARK FAILED.



WARNING:

Use CTX ADM.MARK FAILED with caution. It should only be used as a last resort and only when no other session is touching the index. Normally, CTX ADM.MARK FAILED does not succeed if another session is actively building the index with CREATE or ALTER INDEX. However, index creation or alteration may include windows of time during which CTX ADM. MARK FAILED can succeed, marking the index as failed even as it is being built by another session.

Note:

The background processes used to sync the index for automatic-sync indexes are considered as different sessions and CTX ADMIN.MARK FAILED will not succeed in such scenarios. Run CTX ADM. MARK FAILED only when there are no active background processes.

CTX ADM.MARK FAILED works with local partitioned indexes. However, it changes the status of all partitions to FAILED. Therefore, you should rebuild all index partitions with ALTER INDEX REBUILD PARTITION PARAMETERS ('RESUME') after using CTX ADM.MARK FAILED. If you run ALTER INDEX PARAMETER ('RESUME') after this operation, then Oracle resets the index partition status to valid. Oracle does not rebuild the index partitions that were successfully built before the MARK FAILED operation.

Syntax

```
CTX ADM.MARK FAILED (
  owner_name in VARCHAR2, index_name in VARCHAR2);
```

owner name

The name of the owner of the index whose status is to be changed.

index name

The name of the index whose status is to be changed.



The index name must not be prefixed by the schema or the owner name.



Example

```
begin
   CTX_ADM.MARK_FAILED('owner_1', 'index_1');
end:
```

5.3 RECOVER

The RECOVER procedure cleans up the Text data dictionary, deleting objects such as leftover preferences.

Syntax

```
CTX ADM.RECOVER;
```

Example

```
begin
    ctx_adm.recover;
end;
```

5.4 RESET_AUTO_OPTIMIZE_STATUS

Use the RESET_AUTO_OPTIMIZE_STATUS procedure to reset (or delete the contents of) the CTX AUTO OPTIMIZE STATUS view.

You must log on as CTXSYS to run CTX_ADM.RESET_AUTO_OPTIMIZE_STATUS.

Syntax

```
CTX_ADM.RESET_AUTO_OPTIMIZE_STATUS;
```

Example

```
begin
    ctx_adm.reset_auto_optimize_status;
end;
```

5.5 SET PARAMETER

The SET_PARAMETER procedure sets system-level parameters for index creation and for near real-time indexes.

Syntax

```
CTX_ADM.SET_PARAMETER(param_name IN VARCHAR2, param_value IN VARCHAR2);
```

param_name

Specify the name of the parameter to set, which can be one of the following parameters:

- max index memory (maximum memory allowed for indexing)
- default index memory (default memory allocated for indexing)
- ctx doc key type (default input key type for CTX DOC procedures)
- auto optimize (ENABLE or DISABLE for auto optimization)



- default datastore (default datastore preference)
- default filter file (default filter preference for data stored in files)
- default filter text (default text filter preference)
- default filter binary (default binary filter preference)
- default section html (default html section group preference)
- default section xml (default xml section group preference)
- default section text (default text section group preference)
- default lexer (default lexer preference)
- default wordlist (default wordlist preference)
- default stoplist (default stoplist preference)
- default storage (default storage preference)
- default_ctxcat_lexer (default lexer preference for CTXCAT index)
- default ctxcat stoplist (default stoplist preference for CTXCAT index)
- default ctxcat storage (default CTXCAT index storage
- default ctxcat wordlist (default wordlist preference for CTXCAT index)
- default_ctxrule_lexer (default lexer for CTXRULE index)
- default ctxrule stoplist (default stoplist for CTXRULE index)
- default_ctxrule_storage (default storage for CTXRULE index)
- default ctxrule wordlist (default wordlist for CTXRULE index)

See Also:

To learn more about the default values for these parameters, see "System Parameters" in Oracle Text Indexing Elements

Note:

log_directory (directory for CTX_OUTPUT files) and auto_optimize_logfile (the base file name for the auto optimization log file) can no longer be modified. Any call to the API is ignored for these parameters.

param_value

Specify the value to assign to the parameter. For max_index_memory and default_index_memory, the value you specify must have the following syntax:

number[K|M|G]

where K stands for kilobytes, M stands for megabytes, and G stands for gigabytes. For each of the other parameters, specify the *name* of a preference to use as the default for indexing.



For auto_optimize, the value you specify must be either ENABLE or DISABLE. When you set this parameter to ENABLE, auto optimization jobs can be started. When you set this parameter to DISABLE, no auto optimization jobs can be started and all the currently-running optimization jobs are terminated.

Example

To modify the MAX_INDEX_MEMORY value:

```
exec ctx_adm.set_parameter('MAX_INDEX_MEMORY', 100G);
```

The memory parameter in the indexing statements can be as high as 256 GB (if the MAX INDEX MEMORY parameter is not explicitly specified to a lower value).

```
create index myindex1 on mytab(textcol) indextype is ctxsys.context
parameters ('memory 256G');
exec ctx ddl.sync index(' myindex2', memory=> '256G');
```

Example

```
begin
   ctx_adm.set_parameter('default_lexer', 'my_lexer');
end;
```



6

CTX_ANL Package

This chapter contains the following topics.

- About CTX_ANL Package Procedures
- ADD_DICTIONARY
- DROP_DICTIONARY

6.1 About CTX_ANL Package Procedures

The CTX_ANL PL/SQL package is used with AUTO_LEXER and provides procedures for adding and dropping a custom dictionary from the lexer. A custom dictionary might be one that you develop for a special field of study or for your industry. In most cases, the dictionaries supplied with Oracle Text are more than sufficient to handle your requirements.



"AUTO_LEXER" for a discussion of AUTO_LEXER and supported languages

The CTX ANL package contains the following stored procedures.

Name	Description
ADD_DICTIONARY	Adds a custom dictionary to the lexer.
DROP_DICTIONARY	Drops a custom dictionary from the lexer.



Only the CTXSYS user can use the procedures in CTX ANL.

The APIs in the CTX_ANL package do not support identifiers that are prefixed with the schema or the owner name.

6.2 ADD_DICTIONARY

Use the CTX_ANL.ADD_DICTIONARY procedure to add a custom dictionary to be used by "AUTO LEXER".



The dictionary data is not processed until index/policy creation time or ALTER INDEX time. Errors in dictionary data format are detected at index/policy creation time or ALTER INDEX time and result in error: DRG-13710: Syntax Error in Dictionary.

Syntax

```
CTX_ANL.ADD_DICTIONARY(
name in VARCHAR2,
language in VARCHAR2,
dictionary in CLOB
);
```

name

The unique name for the user-created custom dictionary.



The unique name may not be prefixed by the schema or the owner name as this syntax is not supported.

language

The language used by the custom dictionary.

dictionary

The CLOB containing the custom dictionary. The custom dictionary comprises a list of definitions, which are declared separated by a tab or one per line as described in "Custom Dictionary Format and Syntax".

Custom Dictionary Format and Syntax

The custom dictionary enables you to define a new stem or redefine an existing stem to add words to AUTO_LEXER for your language.

Define a new stem or redefine an existing one using the following syntax:

 $\verb|COMPOUND| < tab> word < tab> STEM < tab> word < tab> parts-of-speech < tab> features < tab| constant < tab$

COMPOUND

Use COMPOUND to create a compound word by joining two whole words with a pipe (|). The word is a simple text string that you want to join to another word to create one compound word to add to the language you specify in AUTO LEXER.

Note that COMPOUND supports a maximum of 8 component words for a compound word.

STEM

Use STEM to add the root for a new word.

word

For COMPOUND and STEM, the word value is a simple text string respresenting a word that you want to join with another word to create a new word; or a word root or stem that you want to add to the language dictionary in AUTO LEXER.



parts-of-speech

The parts-of-speech value is a list of valid parts of speech, separated by a comma. Table 6-1 lists the names for parts-of-speech value. At least one parts-of-speech value is required.

features

The features represent a list of valid linguistic features, as shown in Table 6-2. Multiple features are separated by a comma. Features are optional. If the word is already defined in the supplied language dictionary, then this definition overrides it. It is an error to have an invalid value for parts-of-speech or features.

Table 6-1 Custom Dictionary Valid Parts-of-Speech (case sensitive)

Part-of-Speech	Description
noun	A simple noun, like table, book, or procedure.
nounProper	A proper name, for person, place, etc., typically capitalized, like Zachary, Supidito, Susquehanna
adjective	Modifiers of nouns, which typically can be compared (green, greener, greenest), like fast, trenchant, pendulous.
adverb	Any general modifier of a sentence that may modify an adjective or verb or may stand alone, like slowly, yet, perhaps.
preposition	A word that forms a prepositional phrase with a noun, like off, beside, from. Used for postpositions too, in languages that have postpositions of similar function.

Table 6-2 lists the features and their usage. The specified language determines whether these are relevant and necessary. Note that *declension* refers to the inflection some languages use to determine number (singular or plural), case, and gender. The features are relevant depending on the language for the custom dictionary.

Table 6-2 Custom Dictionary Valid Features

Feature (case sensitive)	Description
genderMasculine	masculine
genderFeminine	feminine
genderNeuter	neuter
declensionHard	hard declension
declensionSoft	soft declension

Examples

```
exec CTX_DDL.CREATE_PREFERENCE('A_LEX', 'AUTO_LEXER');
exec CTX_ANL. ADD_DICTIONARY('my_dict1', 'ENGLISH', lobloc);
select * from CTX_USR_ANL_DICTS;
exec CTX DDL.SET ATTRIBUTE('A LEX', 'english dictionary', 'MY ENGLISH');
```

The following example creates a custom dictionary named d1 to be added to AUTO_LEXER for the English language.

```
declare
dict clob;
begin
dict := '# compounds
COMPOUND help|desk
```



```
COMPOUND
             help|desks
          book|shelf
COMPOUND
COMPOUND
           book|shelves
COMPOUND
             back|woods|man
'# define company abbreviations
STEM
     comp. noun
      ltd.
STEM
     co.
STEM
             noun
     oracle nounProper
STEM
STEM make verb
STEM
     unkword noun
STEM unkword verb
ctx_anl.add_dictionary('d1','ENGLISH',dict);
```

Related Topics

```
"AUTO_LEXER"

"CREATE_PREFERENCE"

"SET_ATTRIBUTE"

"DROP_DICTIONARY"
```

6.3 DROP DICTIONARY

Use this procedure to drop a custom dictionary from AUTO_LEXER.

Syntax

```
CTX_ANL.DROP_DICTIONARY(
name in VARCHAR2,
language in VARCHAR2,
dictionary in CLOB
);
```

name

The unique name for the user-created custom dictionary.



The unique name may not be prefixed by the schema or the owner name as this syntax is not supported.

language

The language for the custom dictionary.

dictionary

The CLOB representing the custom dictionary.



Example

```
begin
   CTX_ANL.DROP_DICTIONARY('dict1', 'english', 'dictionary');
end;
```

Related Topic

"AUTO_LEXER"

"ADD_DICTIONARY"



7

CTX_CLS Package

This chapter contains the following topics.

- About CTX_CLS Package Procedures
- TRAIN
- CLUSTERING

7.1 About CTX_CLS Package Procedures

The CTX_CLS PL/SQL package provides procedures for generating rules that define document categories, and enables you to perform document classification.

The following procedures are in the CTX CLS PL/SQL package.

Name	Description
TRAIN	Generates rules that define document categories. Output based on input training document set.
CLUSTERING	Generates clusters for a document collection.
SA_TRAIN_MODEL	Trains a sentiment classifier.
SA_DROP_MODEL	Drops an existing sentiment classifier.



The APIs in the $\mathtt{CTX_CLS}$ package do not support identifiers that are prefixed with the schema or the owner name.

See Also:

Oracle Text Application Developer's Guide for more information on document classification

7.2 TRAIN

Use this procedure to generate query rules that select document categories. You must supply a training set consisting of categorized documents. Documents can be in any format supported by Oracle Text and must belong to one or more categories. This procedure generates the queries that define the categories and then writes the results to a table.

You must also have a document table and a category table. The category table must contain at least two categories.

For example, your document and category tables can be defined as:

```
create table trainingdoc(
docid number primary key,
text varchar2(4000));

create table category (
docid trainingdoc(docid),
categoryid number);
```

You can use one of two syntaxes depending on the classification algorithm you need. The query compatible syntax uses the RULE_CLASSIFIER preference and generates rules as query strings. The Support Vector Machine syntax uses the SVM_CLASSIFER preference and generates rules in binary format. The SVM_CLASSIFIER is good for high classification accuracy, but because its rules are generated in binary format, they cannot be examined like the query strings generated with the RULE_CLASSIFIER. Note that only those document ids that appear in both the document table and the category table will impact RULE_CLASSIFIER and SVM CLASSIFIER learning.

The CTX_CLS.TRAIN procedure requires that your document table have an associated context index. For best results, the index should be synchronized before running this procedure. SVM_CLASSIFIER syntax enables the use of an unpopulated context index, while query-compatible syntax requires that the context index be populated.



When downgrading the database, you must drop any models that were created in Oracle Database 12c Release 2 (12.2) using TRAIN. These models are not compatible with earlier releases. The following error occurs if the models are not dropped before the downgrade: ORA-40350: One or more models exist that cannot be downgraded.

See Also:

Oracle Text Application Developer's Guide for more on document classification

Query Compatible Syntax

The following syntax generates query-compatible rules and is used with the RULE_CLASSIFIER preference. Use this syntax and preference when different categories are separated from others by several key words. An advantage of generating your rules as query strings is that you can easily examine the generated rules. This is different from generating SVM rules, which are in binary format.

```
cttab in varchar2, catdocid in varchar2, catid in varchar2, restab in varchar2, restab
```



```
rescatid in varchar2,
resquery in varchar2,
resconfid in varchar2,
preference in varchar2 DEFAULT NULL
);
```

index name

Specify the name of the context index associated with your document training set.

docid

Specify the name of the document ID column in the document table. The document IDs in this column must be unique, and this column must be of datatype NUMBER. The values for this column must be stored in an unsigned 32-bit integer and must be in the range 0-4294967295.

cattab

Specify the name of the category table. You must have the READ or SELECT privilege on this table. (See *Oracle Database Security Guide* for information about the READ privilege.)

catdocid

Specify the name of the document ID column in the category table. The document IDs in this table must also exist in the document table. This column must be a NUMBER. The values for this column must be stored in an unsigned 32-bit integer and must be in the range 0-4294967295.

catid

Specify the name of the category ID column in the category table. This column must be a NUMBER. The values for this column must be stored in an unsigned 32-bit integer and must be in the range 0-4294967295.

restab

Specify the name of the result table. You must have INSERT privilege on this table.

rescatid

Specify the name of the category ID column in the result table. This column must be a NUMBER. The values for this column must be stored in an unsigned 32-bit integer and must be in the range 0-4294967295.

resquery

Specify the name of the query column in the result table. This column must be VARACHAR2, CHAR, CLOB, NVARCHAR2, Or NCHAR.

The queries generated in this column connects terms with AND or NOT operators, such as:

```
'T1 & T2 ~ T3'
```

Terms can also be theme tokens and be connected with the ABOUT operator, such as:

```
'about(T1) & about(T2) ~ about(T3)'
```

Generated rules also support WITHIN queries on field sections.

resconfid

Specify the name of the confidence column in result table. This column contains the estimated probability from training data that a document is relevant if that document satisfies the query.

preference

Specify the name of the preference. For classifier types and attributes, see "Classifier Types" in Oracle Text Indexing Elements.



Syntax for Support Vector Machine (SVM) Rules

The Support Vector Machine, or SVM, rules preference generates rules in binary format. Use this syntax when your application requires high classification accuracy.

The following syntax generates Support Vector Machine (SVM) rules with the SVM CLASSIFIER preference.

```
CTX_CLS.TRAIN(

index_name in varchar2,
docid in varchar2,
cattab in varchar2,
catdocid in varchar2,
catid in varchar2,
restab in varchar2,
preference in varchar2);
```

index_name

Specify the name of the text index.

docid

Specify the name of docid column in document table.

cattab

Specify the name of category table.

catdocid

Specify the name of docid column in category table.

catid

Specify the name of category ID column in category table.

restab

Specify the name of result table.

The result table has the following format:

Column Name	Datatype	Description
CAT_ID	NUMBER	The ID of the category.
TYPE	NUMBER(3) NOT NULL	0 for the actual rule or catid; 1 for other.
RULE	BLOB	The returned rule.

preference

Specify the name of user preference. For classifier types and attributes, see "Classifier Types" in Oracle Text Indexing Elements.



Column names must not be prefixed by the owner, schema or table name.

Example

The CTX_CLS.TRAIN procedure is used in supervised classification. For an extended example, see *Oracle Text Application Developer's Guide*.



7.3 CLUSTERING

Use this procedure to cluster a collection of documents. A *cluster* is a group of documents similar to each other in content.

A clustering result set is composed of document assignments and cluster descriptions:

- A document assignment result set shows how relevant each document is to all generated leaf clusters.
- A cluster description result set contains information about what topic a cluster is about.
 This result set identifies the cluster and contains cluster description text, a suggested cluster label, and a quality score for the cluster.

Cluster output is hierarchical. Only leaf clusters are scored for relevance to documents. Producing more clusters requires more computing time. Indicate the upper limit for generated clusters with the <code>CLUSTER_NUM</code> attribute of the <code>KMEAN_CLUSTERING</code> cluster type (see "Cluster Types" in this chapter).

There are two versions of this procedure: one with a table result set, and one with an inmemory result set.

Clustering is also known as unsupervised classification.



For more information about clustering and relevant preferences, see Cluster Types in Oracle Text Indexing Elements, as well as the Oracle Text Application Developer's Guide

Syntax: Table Result Set

index_name

Specify the name of the context index on collection table.

docid

Specify the name of document ID column of the collection table.

doctab_name

Specify the name of document assignment table. This procedure creates the table with the following structure:

```
doc_assign(
   docid number,
   clusterid number,
   score number
);
```



Column	Description
DOCID	Document ID to identify document.
CLUSTERID	ID of a leaf cluster associated with this document. If CLUSTERID is -1, then the cluster contains "miscellaneous" documents; for example, documents that cannot be assigned to any other cluster category.
SCORE	The associated score between the document and the cluster.

If you require more columns, then create the table before you call this procedure.

clstab_name

Specify the name of the cluster description table. This procedure creates the table with the following structure:

```
cluster_desc(
  clusterid NUMBER,
  descript VARCHAR2(4000),
  label VARCHAR2(200),
  sze NUMBER,
  quality_score NUMBER,
  parent NUMBER
```

Column	Description
CLUSTERID	Cluster ID to identify cluster. If CLUSTERID is -1, then the cluster contains "miscellaneous" documents; for example, documents that cannot be assigned to any other cluster category.
DESCRIPT	String to describe the cluster.
LABEL	A suggested label for the cluster.
SZE	This parameter currently has no value.
QUALITY_SCORE	The quality score of the cluster. A higher number indicates greater coherence.
PARENT	The parent cluster ID. Zero means no parent cluster.

If you require more columns, then create the table before you call this procedure.

pref_name

Specify the name of the preference.

Syntax: In-Memory Result Set

Put the result set into in-memory structures for better performance. Two in-memory tables are defined in CTX CLS package for document assignment and cluster description respectively.



index_name

Specify the name of context index on the collection table.

docid

Specify the document ID column of the collection table.

dids

Specify the name of the in-memory docid tab.

```
TYPE docid tab IS TABLE OF number INDEX BY BINARY INTEGER;
```

doctab_name

Specify name of the document assignment in-memory table. This table is defined as follows:

```
TYPE doc_rec IS RECORD (
   docid NUMBER,
   clusterid NUMBER,
   score NUMBER
)
TYPE doc_tab IS TABLE OF doc_rec INDEX BY BINARY_INTEGER;
```

Column	Description
DOCID	Document ID to identify document.
CLUSTERID	ID of a leaf cluster associated with this document. If CLUSTERID is -1, then the cluster contains "miscellaneous" documents; for example, documents that cannot be assigned to any other cluster category.
SCORE	The associated score between the document and the cluster.

cls_tab

Specify the name of cluster description in-memory table.

```
TYPE cluster_rec IS RECORD(
    clusterid NUMBER,
    descript VARCHAR2(4000),
    label VARCHAR2(200),
    sze NUMBER,
    quality_score NUMBER,
    parent NUMBER
);
TYPE cluster_tab IS TABLE OF cluster_rec INDEX BY BINARY_INTEGER;
```

Column	Description
CLUSTERID	Cluster ID to identify cluster. If CLUSTERID is -1, then the cluster contains "miscellaneous" documents; for example, documents that cannot be assigned to any other cluster category.
DESCRIPT	String to describe the cluster.
LABEL	A suggested label for the cluster.
SZE	This parameter currently has no value.
QUALITY_SCORE	The quality score of the cluster. A higher number indicates greater coherence.
PARENT	The parent cluster ID. Zero means no parent cluster.



pref_name

Specify the name of the preference. For cluster types and attributes, see Cluster Types in Oracle Text Indexing Elements.

Example



The Oracle Text Application Developer's Guide for an example of using clustering

7.4 SA_TRAIN_MODEL

Use this procedure to train a sentiment classifier. You must provide a training set consisting of categorized documents to train the sentiment classifier.

Documents can be in any format supported by Oracle Text and must belong to one or more categories.

Oracle Text first validates the training set table and the categories that are provided. Features extracted from the training set documents are used to train the sentiment classifier. A rule table is created and populated with rules that are generated after the sentiment classifier is trained. The sentiment classifier uses these rules to perform sentiment analysis. The CTXRULE index on the rule table is also built.

Note:

When downgrading the database, you must drop any models that were created in Oracle Database 12c Release 2 (12.2) using <code>SA_TRAIN_MODEL</code>. These models are not compatible with earlier releases. The following error occurs if the models are not dropped before the downgrade: ORA-40350: One or more models exist that cannot be downgraded.

Syntax

```
SA_TRAIN_MODEL(
    clsfier_name IN VARCHAR2,
    index_name IN VARCHAR2,
    docid IN VARCHAR2,
    cattab IN VARCHAR2,
    catdocid IN VARCHAR2,
    catid IN VARCHAR2,
    pref_name IN VARCHAR2
);
```

clsfier_name

Specify the name of the sentiment classifier that must be trained. The maximum length of the sentiment classifier name is 24 bytes.

index name

Specify the name of text index associated with the document training set. This is a CONTEXT index that must be created on the training data before the sentiment classifier is trained.

docid

Specify the name of the document ID column in the document training set. The document IDs in this column must be unique, and this column must be of data type NUMBER. The values for this column must be stored in an unsigned 32-bit integer and must be in the range 0 to 4294967295.

cattab

Specify the name of the category table that contains the true labels for the training set documents. This table should contain the <code>docid</code> to <code>catid</code> mappings for training the sentiment classifier.

catdocid

Specify the name of document ID column in the category table. The document IDs in this table must also exist in the document table. This column must be a NUMBER. The values for this column must be stored in an unsigned 32-bit integer and must be in the range 0 to 4294967295.

catid

Specify the name of the category ID column in the category table. This column must be a NUMBER. The values for this column can be either 0, 1, or 2. 0 stands for neutral, 1 stands for positive, and 2 stands for negative.

pref_name

Specify the name of sentiment classifier preference, of type <code>SENTIMENT_CLASSIFIER</code>, which is used to train the sentiment classifier. If no name is provided, then the default sentiment classifier, <code>CTXSYS.DEFAULT SENT CLASSIFIER</code>, is used.

Related Topics

Oracle Text Application Developer's Guide

7.5 SA DROP MODEL

Use this procedure to drop an existing sentiment classifier.

Syntax

```
SA_DROP_MODEL(
    clsfier_name IN VARCHAR2
);
```

clsfier_name

Specify the name of the sentiment classifier that must be dropped.



CTX_DDL Package

The CTX_DDL PL/SQL package provides stored procedures and functions to create and manage the preferences, section groups, and stoplists required for Text indexes.

Name	Description
ADD_ATTR_SECTION	Adds an attribute section to an XML section group.
ADD_AUTO_OPTIMIZE	Adds an index or partition to the list of indexes subject to auto optimization.
ADD_FIELD_SECTION	Creates a field section and assigns it to the specified section group.
ADD_INDEX	Adds an index to a catalog index preference.
ADD_MDATA	Changes the MDATA value of a document.
ADD_MDATA_COLUMN	Maps a FILTER BY column to the specified MDATA section.
ADD_MDATA_SECTION	Adds an MDATA metadata section to a document.
ADD_NDATA_SECTION	Adds an NDATA section to a document.
ADD_SDATA_COLUMN	Maps a FILTER BY column to the specified SDATA section.
ADD_SDATA_SECTION	Adds an SDATA structured data section to a document.
ADD_SEC_GRP_ATTR_VAL	Adds a section group attribute value to the list of values of an already existing section group attribute.
ADD_SPECIAL_SECTION	Adds a special section to a section group.
ADD_STOPCLASS	Adds a stopclass to a stoplist.
ADD_STOP_SECTION	Adds a stop section to an automatic section group.
ADD_STOPTHEME	Adds a stoptheme to a stoplist.
ADD_STOPWORD	Adds a stopword to a stoplist.
ADD_SUB_LEXER	Adds a sub-lexer to a multi-lexer preference.
ADD_ZONE_SECTION	Creates a zone section and adds it to the specified section group.
COPY_POLICY	Creates a copy of a policy.
CREATE_INDEX_SET	Creates an index set for CTXCAT index types.
CREATE_POLICY	Creates a policy to use with ORA: CONTAINS().
CREATE_PREFERENCE	Creates a preference in the Text data dictionary.
CREATE_SECTION_GROUP	Creates a section group in the Text data dictionary.
CREATE_SHADOW_INDEX	Creates a policy for the passed-in index. For nonpartitioned index, also creates an index table.
CREATE_STOPLIST	Creates a stoplist.
DROP_INDEX_SET	Drops an index set.
DROP_POLICY	Drops a policy.
DROP_PREFERENCE	Deletes a preference from the Text data dictionary.
DROP_SECTION_GROUP	Deletes a section group from the Text data dictionary.

Name	Description
DROP_SHADOW_INDEX	Drops a shadow index.
DROP_STOPLIST	Drops a stoplist.
EXCHANGE_SHADOW_INDEX	Swaps the shadow index metadata and data.
LOAD_STOPLIST	
OPTIMIZE_INDEX	Optimizes the index.
POPULATE_PENDING	Populates the pending queue with every rowid in the base table or table partition.
PREFERENCE_IMPLICIT_COMMIT	Specifies whether procedures related to CTX_DDL preferences issue an implicit commit.
RECREATE_INDEX_ONLINE	Recreates the passed-in index.
REM_SEC_GRP_ATTR_VAL	Removes a specific section group attribute value from the list of values of an existing section group attribute.
REMOVE_AUTO_OPTIMIZE	Removes an index or partition from the list of indexes subject to auto optimization
REMOVE_INDEX	Removes an index from a CTXCAT index preference.
REMOVE_MDATA	Removes MDATA values from a document.
REMOVE_SECTION	Deletes a section from a section group.
REMOVE_STOPCLASS	Deletes a stopclass from a stoplist.
REMOVE_STOPTHEME	Deletes a stoptheme from a stoplist.
REMOVE_STOPWORD	Deletes a stopword from a stoplist.
REMOVE_SUB_LEXER	Deletes a sub-lexer from a multi-lexer preference.
REPLACE_INDEX_METADATA	Replaces metadata for local domain indexes.
SET_ATTRIBUTE	Sets a preference attribute.
SET_SEC_GRP_ATTR	Adds a section group-specific attribute to a section group identified by name.
SET_SECTION_ATTRIBUTE	Sets a section attribute.
SYNC_INDEX	Synchronizes the index.
UNSET_ATTRIBUTE	Removes a set attribute from a preference.
UPDATE_SUB_LEXER	Updates a sub-lexer.
UNSET_SEC_GRP_ATTR	Removes a section group specific attribute.
UPDATE_POLICY	Updates a policy.
UPDATE_SDATA	Updates an SDATA section.

Note:

Except CREATE_PREFERENCE and CREATE_SECTION_GROUP, the APIs in the CTX_DDL package do not support identifiers that are prefixed with the schema or owner name.

8.1 ADD_ATTR_SECTION

Adds an attribute section to an XML section group. This procedure is useful for defining attributes in XML documents as sections. This enables you to search XML attribute text with the WITHIN operator.



When you use AUTO_SECTION_GROUP, attribute sections are created automatically. Attribute sections created automatically are named in the form tag@attribute.

Syntax

```
CTX_DDL.ADD_ATTR_SECTION(
group_name IN VARCHAR2,
section_name IN VARCHAR2,
tag IN VARCHAR2);
```

group_name

Specify the name of the XML section group. You can add attribute sections only to XML section groups.

section_name

Specify the name of the attribute section. This is the name used for WITHIN queries on the attribute text

The section name you specify cannot contain the colon (:), comma (,), or dot (.) characters.

The section name must also be unique within <code>group_name</code>. Section names are case-insensitive.

Attribute section names can be no more than 64 bytes long.

tag

Specify the name of the attribute in tag@attr form. This parameter is case-sensitive.

Examples

Consider an XML file that defines the BOOK tag with a TITLE attribute as follows:

```
<BOOK TITLE="Tale of Two Cities">
  It was the best of times.
</BOOK>
```

To define the title attribute as an attribute section, create an XML_SECTION_GROUP and define the attribute section as follows:

```
begin
ctx_ddl.create_section_group('myxmlgroup', 'XML_SECTION_GROUP');
ctx_ddl.add_attr_section('myxmlgroup', 'booktitle', 'BOOK@TITLE');
end:
```

When you define the TITLE attribute section as such and index the document set, you can query the XML attribute text as follows:

```
'Cities within booktitle'
```



Related Topic

"PREFERENCE IMPLICIT COMMIT"

8.2 ADD_AUTO_OPTIMIZE

Adds an index or partition to the list of indexes subject to auto optimization. For partitioned indexes, the name of the partition must be specified, or else an error occurs. For global indexes, STAGE ITAB must be enabled, or else an error occurs.



In Oracle Database Release 21c, the procedures <code>ADD_AUTO_OPTIMIZE</code> and <code>REMOVE_AUTO_OPTIMIZE</code>, and the views <code>CTX_AUTO_OPTIMIZE_INDEXES</code>, <code>CTX_USER_AUTO_OPTIMIZE_INDEXES</code> and <code>CTX_AUTO_OPTIMIZE_STATUS</code> are deprecated.

The AUTO_OPTIMIZE feature improves the manageability of indexes that use the STAGE_ITAB feature. The STAGE_ITAB feature introduces a staging \$G table to collect postings from newly synced documents.

The AUTO OPTIMIZE feature has the following goals:

- Enables you to register indexes and partitions to a background AUTO OPTIMIZE process.
- Automatically moves rows from the \$G table to \$I at appropriate times.
- Movement of rows from \$G to \$I is done in a way to maximize query performance.

This procedure starts the background process if it has not already been started. The progress of the auto optimization is tracked by CTX logging.

The changes made by this procedure take effect immediately.



The init.ora parameter <code>JOB_QUEUE_PROCESSES</code> must be set to one or higher. See Oracle Database Reference for more information about <code>JOB_QUEUE_PROCESSES</code>.

Syntax

idx_name

Specify the name of the index to add.

part_name

Specify the name of the partition to add.



optlevel

Specifies the optlevel of the CTX_DDL.OPTIMIZE_INDEX procedure. The only valid value for this parameter is merge.

Notes

The recommended sequence of steps for using auto optimization is:

- Create the required indexes.
- Add these indexes to the auto optimization list by using the CTX_DDL.ADD_AUTO_OPTIMIZE procedure.

The synchronize index operation automatically begins executing an auto optimization job (unless it is already running). This job continues until it runs out of work. Future synchronize index operations will automatically start executing the auto optimization job, if it is not already running.

Related Topics

```
"REMOVE_AUTO_OPTIMIZE"
```

Oracle Text Application Developer's Guide for information about using STAGE_ITAB with CONTEXT indexes

SYNC_INDEX

8.3 ADD_FIELD_SECTION

Creates a field section and adds the section to an existing section group. This enables field section searching with the WITHIN operator. You can add an unlimited number of field sections.

Field sections are delimited by start and end tags. By default, the text within field sections are indexed as a sub-document separate from the rest of the document.

Unlike zone sections, field sections cannot nest or overlap. As such, field sections are best suited for non-repeating, non-overlapping sections such as <code>TITLE</code> and <code>AUTHOR</code> markup in email- or news-type documents.

Because of how field sections are indexed, WITHIN queries on field sections are usually faster than WITHIN queries on zone sections.

Syntax

```
CTX_DDL.ADD_FIELD_SECTION(
  group_name IN VARCHAR2,
  section_name IN VARCHAR2,
  tag IN VARCHAR2,
  visible IN BOOLEAN default FALSE
);
```

group name

Specify the name of the section group to which <code>section_name</code> is added. You can add an unlimited number of field sections to a single section group. Within the same group, section zone names and section field names cannot be the same.

section name

Specify the name of the section to add to the <code>group_name</code>. Use this name to identify the section in queries. Avoid using names that contain non-alphanumeric characters such as _, because these characters must be escaped in queries. Section names are case-insensitive.



The section_name may not be prefixed by the schema or the owner name as this syntax is not supported.

Within the same group, zone section names and field section names cannot be the same. The terms *Paragraph* and *Sentence* are reserved for special sections.

Section names need not be unique across tags. You can assign the same section name to more than one tag, which makes details transparent to searches.

tag

Specify the tag that marks the start of a section. For example, if the tag is <H1>, then specify H1. The start tag you specify must be unique within a section group.



The tag may not be prefixed by the schema or the owner name as this syntax is not supported.

If group_name is an HTML_SECTION_GROUP, then you can create field sections for the META tag's NAME/CONTENT attribute pairs. To do so, specify tag as meta@namevalue where namevalue is the value of the NAME attribute whose CONTENT attribute is to be indexed as a section. Refer to the example "Creating Sections for <META> Tags".

Oracle Text knows what the end tags look like from the <code>group_type</code> parameter you specify when you create the section group.

visible

Specify TRUE to make the text visible within the rest of the document.

By default the <code>visible</code> flag is <code>FALSE</code>. This means that Oracle Text indexes the text within field sections as a sub-document separate from the rest of the document. However, you can set the visible flag to <code>TRUE</code> if you want text within the field section to be indexed as part of the enclosing document.

Examples

Visible and Invisible Field Sections

The following example defines a section group basicgroup of the BASIC_SECTION_GROUP type. (See "Section Group Types" for information about the BASIC_SECTION_GROUP type.) The example then creates a field section in basicgroup called Author for the <A> tag.

The example also sets the visible flag to FALSE:

```
begin

ctx_ddl.create_section_group('basicgroup', 'BASIC_SECTION_GROUP');
ctx_ddl.add_field_section('basicgroup', 'Author', 'A', FALSE);
end;
```



Because the Author field section is not visible, to find text within the Author section, you must use the WITHIN operator as follows:

```
'(Martin Luther King) WITHIN Author'
```

A query of *Martin Luther King* without the WITHIN operator does not return instances of this term in field sections. To query text within field sections without specifying WITHIN, you must set the visible flag to TRUE when you create the section as follows:

```
begin
ctx_ddl.add_field_section('basicgroup', 'Author', 'A', TRUE);
end;
```

Creating Sections for <META> Tags

When you use the HTML SECTION GROUP, you can create sections for META tags.

Consider an HTML document that has a META tag as follows:

```
<META NAME="author" CONTENT="ken">
```

To create a field section that indexes the CONTENT attribute for the <META NAME="author"> tag:

```
begin
ctx_ddl.create_section_group('myhtmlgroup', 'HTML_SECTION_GROUP');
ctx_ddl.add_field_section('myhtmlgroup', 'author', 'META@AUTHOR');
end
```

After indexing with section group mygroup, query the document as follows:

```
'ken WITHIN author'
```

Limitations

Nested Sections

Field sections cannot be nested. For example, if you define a field section to start with <TITLE> and define another field section to start with <FOO>, the two sections cannot be nested as follows:

```
<TITLE> dog <FOO> cat </FOO> </TITLE>
```

To work with nested section define them as zone sections.

Repeated Sections

Repeated field sections are allowed, but WITHIN queries treat them as a single section. The following is an example of repeated field section in a document:

```
<TITLE> cat </TITLE> <TITLE> dog </TITLE>
```

The query (dog and cat) within title returns the document, even though these words occur in different sections.

To have WITHIN queries distinguish repeated sections, define them as zone sections.

Related Topics

"WITHIN"

"Section Group Types"



```
"CREATE_SECTION_GROUP"

"ADD_ZONE_SECTION"

"ADD_SPECIAL_SECTION"

"REMOVE_SECTION"

"DROP_SECTION_GROUP"
```

8.4 ADD INDEX

Use this procedure to add a subindex to a catalog index preference. Create this preference by naming one or more columns in the base table.

Because you create subindexes to improve the response time of structured queries, the column you add should be used in the <code>structured_query</code> clause of the <code>CATSEARCH</code> operator at query time.

Note:

The Oracle Text indextype CTXCAT is deprecated with Oracle Database 23ai. The indextype itself, and it's operator CTXCAT, can be removed in a future release. Both CTXCAT and the use of CTXCAT grammar as an alternative grammar for CONTEXT queries is deprecated. Instead, Oracle recommends that you use the CONTEXT indextype, which can provide all the same functionality, except that it is not transactional. Near-transactional behavior in CONTEXT can be achieved by using SYNC (ON COMMIT) or, preferably, SYNC (EVERY [time-period]) with a short time period.

CTXCAT was introduced when indexes were typically a few megabytes in size. Modern, large indexes, can be difficult to manage with CTXCAT. The addition of index sets to CTXCAT can be achieved more effectively by the use of FILTER BY and ORDER BY columns, or SDATA, or both, in the CONTEXT indextype. CTXCAT is therefore rarely an appropriate choice. Oracle recommends that you choose the more efficient CONTEXT indextype.

Syntax

set name

Specify the name of the index set.

column list

Specify a comma-delimited list of columns to index. At index time, any column listed here cannot have a NULL value in any row in the base table. If any row is NULL during indexing, then an error is raised.

Always ensure that your columns have non-NULL values before and after indexing.





A column name in column_list must not be prefixed by the owner, schema or table name.

storage_clause

Specify a storage clause.

Example

Consider a table called AUCTION with the following schema:

```
create table auction(
  item_id number,
  title varchar2(100),
  category_id number,
  price number,
  bid close date);
```

Assume that queries on the table involve a mandatory text query clause and optional structured conditions on category id. Results must be sorted based on bid close.

You can create a catalog index to support the different types of structured queries a user might enter.

To create the indexes, first create the index set preference then add the required indexes to it:

```
begin
  ctx_ddl.create_index_set('auction_iset');
  ctx_ddl.add_index('auction_iset','bid_close');
  ctx_ddl.add_index('auction_iset','category_id, bid_close');
end;
```

Create the combined catalog index with CREATE INDEX as follows:

```
create index auction_titlex on AUCTION(title) indextype is CTXCAT parameters
('index set auction iset');
```

Querying

To query the title column for the word *pokemon*, enter regular and mixed queries as follows:

```
select * from AUCTION where CATSEARCH(title, 'pokemon', NULL) > 0;
select * from AUCTION where CATSEARCH(title, 'pokemon', 'category_id=99 order by
bid_close desc') > 0;
```

Notes

VARCHAR2 columns in the column list of a CTXCAT index of an index set cannot exceed 30 bytes.

Related Topics

REMOVE_INDEX

Removes the index with the specified column list from a \mathtt{CTXCAT} index set preference.



8.5 ADD_MDATA

Use this procedure to change the metadata of a document that has been specified as an MDATA section.

After this call, MDATA queries involving the named MDATA value will find documents with the given MDATA value.

There are two versions of CTX_DDL.ADD_MDATA: one for adding a single metadata value to a single rowid, and another for handing multiple values, multiple rowids, or both.

CTX_DDL.ADD_MDATA is transactional; it takes effect immediately in the calling session, can be seen only in the calling session, can be reversed with a ROLLBACK command, and must be committed to take permanent effect.

Syntax

This is the syntax for adding a single value to a single rowid:

idx name

Name of the text index that contains the named rowid.

section name

Name of the MDATA section.

mdata_value

The metadata value to add to the document.

mdata rowid

The rowid to which to add the metadata value.

[part_name]

Name of the index partition, if any. Must be provided for local partitioned indexes and must be NULL for global, nonpartitioned indexes.

This is the syntax for handling multiple values, multiple rowids, or both. This version is more efficient for large numbers of new values or rowids.

idx name

Name of the text index that contains the named rowids.



section name

Name of the MDATA section.

mdata values

List of metadata values. If a metadata value contains a comma, the comma must be escaped with a backslash.

mdata rowids

The rowids to which to add the metadata values.

[part_name]

Name of the index partition, if any. Must be provided for local partitioned indexes and must be NULL for global, nonpartitioned indexes.

Restrictions and Limitations

- Use CTX_DDL.REMOVE_MDATA to remove metadata values from already-indexed documents.

 Only the owner of the index is allowed to call ADD MDATA and REMOVE MDATA.
- The CTX DDL.ADD MDATA queries can be slower when using a PDB lockdown profile.
- If a rowid is not yet indexed, CTX_DDL.ADD_MDATA completes without error, but an error is logged in CTX_USER_INDEX_ERRORS.
- These updates are updates directly on the index itself, not on the actual contents stored in the base table. Therefore, they will not exist when the Text index is rebuilt.
- CTX_DDL.ADD_MDATA is not supported for documents with Oracle Text search index as stage itab is ON by default for Oracle Text search index.

Examples

This example updates a single value:

```
select rowid from mytab where contains(text, 'MDATA(sec, value')>0;
No rows returned
exec ctx_ddl.add_mdata('my_index', 'sec', 'value', 'ABC');
select rowid from mytab where contains(text, 'MDATA(sec, value')>0;
ROWID
-----
ABC
```

This example updates multiple values:

This is equivalent to:

```
begin
ctx_ddl.add_mdata('my_index', 'sec', 'value1', 'ABC');
ctx_ddl.add_mdata('my_index', 'sec', 'value1', 'DEF');
ctx_ddl.add_mdata('my_index', 'sec', 'value2', 'ABC');
ctx_ddl.add_mdata('my_index', 'sec', 'value2', 'DEF');
ctx_ddl.add_mdata('my_index', 'sec', 'value3', 'ABC');
ctx_ddl.add_mdata('my_index', 'sec', 'value3', 'DEF');
end;
```



Related Topics

- MDATA
- ADD_MDATA_SECTION
- REMOVE MDATA

Use this procedure to remove metadata values, which are associated with an ${\tt MDATA}$ section, from a document.

Oracle Text Application Developer's Guide

8.6 ADD_MDATA_COLUMN

Use this procedure to map the FILTER BY column named in *column_name* to the MDATA section named in section_name.

Syntax

The syntax is as follows:

group_name

Name of the group that contains the section.

section name

Name of the MDATA section.

column name

Name of the FILTER BY column to add to the MDATA section.



The column name must not be prefixed by the owner, schema or table name.

Restrictions

MDATA sections that are created with CTX_DDL.ADD_MDATA_COLUMN cannot have their values changed using CTX_DDL.ADD_MDATA or CTX_DDL.REMOVE_MDATA. Doing so will result in errors being returned. The section values must be updated using SQL.

Notes

- The stored datatype for MDATA sections is text. Therefore, the value of the FILTER BY column is converted to text during indexing. For non-text datatypes, the FILTER BY columns are normalized to an internal format during indexing. If the section is queried with an MDATA operator, then the MDATA query string will also be normalized to the internal format before processing.
- When a FILTER BY column is mapped as MDATA, the cost-based optimizer in Oracle Text tries to avoid using the Oracle Text composite domain index to process range predicate(s) on that FILTER BY column. This is because range predicates on MDATA FILTER BY columns

are processed less efficiently than if they were declared as SDATA. For this reason, you should not add a FILTER BY column as MDATA if you plan to do range searches on the column.

Related Topics

```
"MDATA"

"ADD_MDATA_SECTION"

"REMOVE_MDATA"

"ADD_SDATA_COLUMN"
```

See Also:

Chapter 8, "Searching Document Sections in Oracle Text" in *Oracle Text Application Developer's Guide*

8.7 ADD_MDATA_SECTION

Use this procedure to add an MDATA section, with an accompanying value, to an existing section group. MDATA sections cannot be added to Null Section groups, Path Section groups, or Auto Section groups.

Section values undergo a simplified normalization:

- Leading and trailing whitespace on the value is removed.
- The value is truncated to 255 bytes.
- The value is indexed as a single value; if the value consists of multiple words, it is not broken up.
- Case is preserved. If the document is dynamically generated, then implement case-insensitivity by uppercasing MDATA values and making sure to search only in uppercase.

Use CTX DDL.REMOVE SECTION to remove sections.

Syntax

group_name

Name of the section group that will contain the MDATA section.

section_name

Name of the MDATA section.

tag

The value of the MDATA section. For example, if the section is <AUTHOR>, the value could be *Cynthia Kadohata* (author of the novel *The Floating World*). More than one *tag* can be assigned to a given MDATA section.

read only

FALSE (default) if you want to allow calling CTX_DDL.ADD_MDATA() and CTX_DDL.REMOVE_MDATA() for this MDATA section, and TRUE otherwise. When set to FALSE, the queries on the MDATA section run less efficiently because a cursor needs to be opened on the index table to track the deleted values for that MDATA section.

Example

This example creates an MDATA section called auth.

```
ctx_ddl.create_section_group('htmgroup', 'HTML_SECTION_GROUP');
ctx ddl.add mdata section('htmgroup', 'auth', 'author', READ ONLY);
```

Related Topics

```
"ADD_MDATA"
"REMOVE_MDATA"
"MDATA"
```

"CREATE_SECTION_GROUP"



Chapter 8, "Searching Document Sections in Oracle Text" in *Oracle Text Application Developer's Guide*

8.8 ADD_NDATA_SECTION

Use this procedure to find matches that are spelled in a similar way. The value of an NDATA section is extracted from the document text like other sections, but is indexed as name data. NDATA sections are stored in the CTX USER SECTIONS view.

Syntax

group_name

Name of the group that contains the section.

section name

Name of the NDATA section.

taq

Name of the tag that marks the start of a section. For example, if the tag is <H1>, specify H1. The start tag you specify must be unique within a section group.

Notes

NDATA sections support both single and multi-byte data, however, there are character- and term-based limitations. NDATA section data that is indexed is constrained as follows:

number of characters in a single, white space delimited term

511

number of white space delimited terms

255

total number of characters, including white spaces

511

NDATA section data that exceeds these constraints are truncated.

Example

The following example defines a section group namegroup of the BASIC_SECTION_GROUP type. It then creates an NDATA section in namegroup called firstname.

```
begin
   ctx_ddl.create_section_group('namegroup', 'BASIC_SECTION_GROUP');
   ctx_ddl.add_ndata_section('namegroup', 'firstname', 'fname1');
end;
```

8.9 ADD_PATH

Adds a path to a path-list preference for use with the PATHLIST parameter, when creating a path-subsetting JSON search index.

You use this parameter with the PL/SQL subprogram CTX_DDL.CREATE_PATH_LIST to specify a subset of paths to include or exclude from indexing. In this way, you can choose to index only relevant paths for efficient search. The excluded fields are not indexed, and the JSON search index is not used for them when querying. Filtering out irrelevant paths from documents can reduce the amount of data indexed, thereby minimizing disk space, indexing costs, and the index creation or rebuild time.

Syntax

pref_name

Specify the name of the pathlist preference to which you want to add a subset of paths.

path_type

Specify the type of search to target for your JSON search index:

- NUMBER: For numeric-value range search
- TIMESTAMP: For date and time value range-search
- VARCHAR2: For string-value range search
- TEXT: For full-text and string-equality search



path_string

Specify a path to add to the specified pathlist preference.



Currently, you cannot add multiple paths in a single path string. To add multiple paths, you must use the ADD PATH parameter to specify each path.

For example:

```
BEGIN
  CTX_DDL.create_path_list('json_pl_incl', CTX_DDL.PATHLIST_JSON,
CTX_DDL.PATHLIST_INCLUDE);
  CTX_DDL.add_path('json_pl_incl', 'TEXT', '<path1>');
  CTX_DDL.add_path('json_pl_incl', 'TEXT', '<path2>');
  CTX_DDL.add_path('json_pl_incl', 'NUMBER', '<path3>');
  END;
//
```

Examples

To create an INCLUDE pathlist:

Here, you first create a list of paths (json_pl_incl) to be included for indexing using the CTX_DDL.CREATE_PATH_LIST and CTX_DDL.ADD_PATH APIs. You then create an index using the PATHLIST parameter, whose value is a named list of the paths to be included.

This example creates a JSON search index for full-text and string-equality searches of fields \$.SpecialInstructions and \$.LineItems.Part.Description of a purchase order document. But it also indexes fields \$.PONumber and \$.LineItems.Part.UnitPrice for numeric-value ranges, and fields \$.Reference, \$.User, \$.ShippingInstructions.name, and \$.ShippingInstructions.Address.zipCode for string-value ranges.

```
-- create a pathlist preference and add paths
BEGIN
  CTX DDL.create path_list('json_pl_incl', CTX DDL.PATHLIST JSON,
CTX DDL.PATHLIST INCLUDE);
  CTX DDL.add path('json pl incl', 'TEXT',
                                                '$.SpecialInstructions');
  CTX DDL.add_path('json pl incl', 'TEXT',
'$.LineItems.Part.Description');
  CTX DDL.add path('json pl incl', 'NUMBER',
                                               '$.PONumber');
  CTX DDL.add path('json pl incl', 'NUMBER',
'$.LineItems.Part.UnitPrice');
  CTX DDL.add_path('json pl incl', 'VARCHAR2', '$.Reference');
  CTX DDL.add path('json pl incl', 'VARCHAR2', '$.User');
  CTX DDL.add_path('json pl incl', 'VARCHAR2',
'$.ShippingInstructions.name');
  CTX DDL.add path('json pl incl', 'VARCHAR2',
'$.ShippingInstructions.Address.zipCode');
END;
-- declare the pathlist preference name
```

```
CREATE SEARCH INDEX json_idx ON json_tab (purchase_order_jsondoc)
FOR JSON PARAMETERS ('PATHLIST json pl incl');
```

To create an EXCLUDE pathlist:

Here, you first create a list of paths (json_pl_excl) to be excluded from indexing using the CTX_DDL.CREATE_PATH_LIST and CTX_DDL.ADD_PATH APIs. You then create an index using the PATHLIST parameter, whose value is a named list of the paths to be excluded.

```
-- create a pathlist preference and add paths
BEGIN
  CTX DDL.create path list('json pl excl', CTX DDL.PATHLIST JSON,
CTX DDL.PATHLIST EXCLUDE);
  CTX DDL.add path('json pl excl', 'TEXT',
                                                '$.SpecialInstructions');
  CTX DDL.add path('json pl excl', 'TEXT',
'$.LineItems.Part.Description');
 CTX DDL.add path('json pl excl', 'NUMBER',
                                               '$.PONumber');
  CTX DDL.add path('json pl excl', 'NUMBER',
'$.LineItems.Part.UnitPrice');
  CTX DDL.add_path('json pl excl', 'VARCHAR2', '$.Reference');
  CTX DDL.add path('json pl excl', 'VARCHAR2', '$.User');
 CTX DDL.add_path('json pl excl', 'VARCHAR2',
'$.ShippingInstructions.name');
 CTX DDL.add path('json pl excl', 'VARCHAR2',
'$.ShippingInstructions.Address.zipCode');
END;
-- declare the pathlist preference name
CREATE SEARCH INDEX json idx ON json tab (purchase order jsondoc)
  FOR JSON PARAMETERS ('PATHLIST json pl excl');
```

Related Topics

CREATE PATH LIST

Creates a path-list preference to use with the PATHLIST parameter, when creating a path-subsetting JSON search index.

DROP_PATH_LIST
 Removes an existing pathlist preference.

8.10 ADD SDATA COLUMN

Use this procedure to map the FILTER BY or ORDER BY column (named in *column_name*) to the SDATA section (named in *section_name*). By default, all FILTER BY columns are mapped as SDATA.

Syntax

The syntax is as follows:

group_name

Name of the group that contains the section.

section name

Name of the SDATA section.

column name

Name of the FILTER BY column to add to the SDATA section.

Usage Notes

- Mapping FILTER BY columns to sections is optional. If no section mapping exists for a FILTER BY column, then it is mapped to an SDATA section, and the section name will be the name of the FILTER BY column.
- If a section group is not specified during CREATE INDEX of a composite domain index, then
 system default section group settings is used, and a SDATA section is created for each of
 the FILTER BY and ORDER BY columns.



Because section name does not allow certain special characters and is case insensitive, if the column name is case sensitive or contains special characters, then an error is raised. To work around this problem, you need to map the column to an MDATA or SDATA section before creating the index.

- An error will be raised if a column mapped to MDATA also appears in the ORDER BY column clause.
- Column section names are unique to their section group. That is, you cannot have an MDATA column section named FOO if you already have an MDATA column section named FOO. Furthermore, you cannot have a field section named FOO if you already have an SDATA column section named FOO. This is true whether it is implicitly created (by CREATE INDEX for FILTER BY OF ORDER BY clauses) or explicitly created (by CTX DDL.ADD SDATA COLUMN).
- One section name can only be mapped to one FILTER BY column, and vice versa. For
 example, mapping a section to more than one column or mapping a column to more than
 one section is not allowed.
- Column sections can be added to any type of section group, including the NULL section group.
- 99 is the maximum number for SDATA sections and columns.
- If the datatype of a FILTER BY or ORDER BY column is DATE, then the DATE datatype values must conform to the YYYY-MM-DD or YYYY-MM-DD HH24:MI:SS format. For example, to store a DATE value of "Nov. 24, 2006 10:32 PM 36 sec", the document appears as <TAG>2006-11-24 22:32:36</TAG>.

Related Topics

SDATA

Use the SDATA operator to perform tests on SDATA sections and columns, which contain structured data values.

ADD SDATA SECTION

This procedure adds an SDATA section to a section group. By default, all FILTER BY columns are mapped as SDATA.

UPDATE SDATA

UPDATE_SDATA is an index API that modifies the specified SDATA values in the index. This API does not store or modify column values in a base table, where the base table column may have been used as an SDATA section.

- ADD MDATA COLUMN
- Oracle Text Application Developer's Guide

8.11 ADD_SDATA_SECTION

This procedure adds an SDATA section to a section group. By default, all FILTER BY columns are mapped as SDATA.

Starting with Oracle Database 12c Release 2 (12.2), searchable multi-valued SDATA sections are supported. There is no restriction on the number of SDATA sections that can be created for an index. That is, the sum total of SDATA sections for an index, created implicitly with FILTER BY and ORDER BY, and explicitly with the CTX_DDL.ADD_SDATA_SECTION() procedure is not restricted anymore. The total number of CDI, including FILTER BY and ORDER BY is 32, but the number of SDATA sections supported is unlimited.

There are two types of SDATA sections:

- Searchable: Creates optimized for search SDATA sections which support multiple values per document for the section and efficient range search capability.
- Sortable: Creates optimized for sort SDATA sections which support a single value per document for the section. If the <code>optimized_for</code> attribute is not set, then the default type of section is Sortable. The Composite Domain Index uses Sortable SDATA internally for efficient <code>FILTER BY or ORDER BY evaluation</code>.

Starting with Oracle Database Release 18c, group counts or facets are supported for SDATA sections that are created with the <code>optimized_for</code> attribute set to <code>sort</code>, <code>search</code>, or <code>sort_and_search</code>. The <code>optimized_for</code> attribute can be set by using the <code>CTX_DDL.SET_SECTION_ATTRIBUTE</code> procedure.

Syntax

group_name

Name of the group that contains the section.

section name

Name of the SDATA section.

tao

Name of the tag to add to the SDATA section.

datatype

Specifies the stored format for the data, as well as the semantics of comparison in later use in SDATA operators.

The supported datatypes for searchable SDATA sections are:

 VARCHAR2: Stores up to 249 bytes of character data in the database character set. Values larger than this result in a per-document indexing error.

Note that leading and trailing whitespace are always trimmed from SDATA section values when extracted by the sectioner. This is different than SDATA columns. Column values are never trimmed. No lexing is performed on the value from either kind of SDATA.

- NUMBER: Stores numeric literals.
- DATE: The DATE datatype values must conform to the YYYY-MM-DD or YYYY-MM-DD HH24:MI:SS format. For example, to store a DATE value of Nov. 24, 2006 10:32 pm 36 sec, the document appears as:

```
<TAG>2006-11-24 22:32:36</TAG>
```

- BINARY FLOAT: Stores 32-bit floating point number.
- BINARY DOUBLE: Stores 64-bit floating point number.
- TIMESTAMP: The TIMESTAMP datatype is an extension of the DATE datatype.

It stores year, month, and day values of date, as well as hour, minute, and second values of time. It also stores fractional seconds, which are not stored by the DATE datatype. The fractional seconds precision cannot be more than 9.

The TIMESTAMP values must follow the ISO format. You can specify the TIMESTAMP literal in the YYYY-MM-DDTHH:MI:SS format. For example:

```
<TAG>1997-11-05T19:20:00</TAG>
```

• TIMESTAMP_WITH_TIMEZONE: The TIMESTAMP_WITH_TIMEZONE datatype is a variant of TIMESTAMP datatype that includes a time zone offset or a time zone region name in its value.

The fractional seconds precision cannot be more than 9. The <code>TIMESTAMP_WITH_TIMEZONE</code> values must follow the ISO format. For example:

```
<TAG>1997-12-31T19:20:00-05:00</TAG>
```



The Searchable SDATA sections do not support the CHAR and RAW datatypes.

The supported datatypes for sortable SDATA sections are:

- VARCHAR2: Stores up to 249 bytes of character data in the database character set. Values larger than this result in a per-document indexing error.
 - Note that leading and trailing whitespace are always trimmed from SDATA section values when extracted by the sectioner. This is different than SDATA columns. Column values are never trimmed. No lexing is performed on the value from either kind of SDATA.
- CHAR: Stores up to 249 bytes of character data in the database character set. Values larger than this result in a per-document indexing error.



Note that leading and trailing whitespace are always trimmed from SDATA section values when extracted by the sectioner. This is different than SDATA columns. Column values are never trimmed. No lexing is performed on the value from either kind of SDATA. To be consistent with SQL, the comparisons of CHAR datatype SDATA values are blank-padded comparisons.

RAW: Stores up to 249 bytes of binary data. Values larger than this result in a per-document indexing error.

The value is converted from hexadecimal string representation. That is, to store a value of 65, the document appears as <TAG>40</TAG>, and not <TAG>65</TAG> or <TAG>A</TAG>.

- NUMBER: Stores numeric literals.
- DATE: The DATE datatype values must conform to the YYYY-MM-DD or YYYY-MM-DD HH24:MI:SS format. For example, to store a DATE value of Nov. 24, 2006 10:32 pm 36 sec, the document appears as:

```
<TAG>2006-11-24 22:32:36</TAG>
```

Note:

The Sortable SDATA sections support the TIMESTAMP datatype when specified using the sdata name in hitlist. In this case, you must explicitly specify the TIMESTAMP datatype using the optimized_for attribute for search or sort_and_search attribute values. A detailed example on this is given at the end of this section.

Limitations

- If no SDATA tag occurs in a given document, then this is treated as an SDATA value of NULL.
- Empty SDATA tags are treated as NULL values.
- SDATA sections cannot be nested. Sections that are nested inside are ignored.
- SDATA sections do not support skipjoins and printjoins characters.

Examples

To create and query a Searchable SDATA section:

The following statements create a table named tab with two rows of data:

```
create table tab(id number, info varchar2(100));
insert into tab values(1,'Hello World<fruit>apple</fruit><price>3</price>');
insert into tab values(2,'Hello World<fruit>orange</fruit><price>5</price>');
```

The following statements create a basic section group named sg, add SDATA sections to it and mark the SDATA to be searchable:

```
exec ctx_ddl.create_section_group('sg', 'basic_section_group');
exec ctx_ddl.add_sdata_section('sg','fruit','fruit','varchar2');
exec ctx_ddl.set section attribute('sg','fruit','optimized for','search');
```

```
exec ctx_ddl.add_sdata_section('sg','price','price','number');
exec ctx ddl.set section attribute('sg','price','optimized for','search');
```

The following statement creates an index on sg:

```
create index idx on tab(info) indextype is ctxsys.context parameters
('section group sg');
```

The following statements query tab to demonstrate searchable SDATA:

Query 1:

```
select id from tab where CONTAINS(info, 'SDATA(fruit = "apple")'); return
id 1
```

Query 2:

```
select id from tab where CONTAINS(info, 'Hello and SDATA(price > 4)'); return id 2
```

To create and query a Sortable SDATA section:

Here, you can see how to use the timestamp data type with sdata name in hitlist for Sortable SDATA sections.

The following statements create a table named t1:

```
create table t1(text varchar2(100));
insert into t1 values('Oracle1 <TAG1>1997-01-31T09:26:50.12</TAG1>');
```

The following statements create a basic section group named sg, add an SDATA section (sec01) to it, and mark the SDATA section to be sortable with the timestamp data type and the optimized for attribute:

```
exec ctx_ddl.create_section_group('sg','basic_section_group');
exec ctx_ddl.add_sdata_section('sg','sec01','tag1','timestamp');
exec ctx_ddl.set_section_attribute('sg','sec01','optimized_for','sort');
```

The following statement creates an index idx1 on sq:

```
create index idx1 on t1(text)
    indextype is ctxsys.context parameters('section group sg');
```

The following statement queries t1 to demonstrate sortable SDATA. Here, you specify the SDATA section (sec01) in hitlist using sdata name:

```
select * from ctx_user_index_errors;
set long 32000
set pagesize 0
variable displayrs clob;
declare
```



```
rs clob;
begin
  ctx_query.result_set('idx1','Oracle1',
'<ctx_result_set_descriptor>
  <count/>
  <hitlist start_hit_num="1" end_hit_num="10">
  <score/>
  <sdata name="sec01"/>
  </hitlist>
  </ctx_result_set_descriptor>',rs);
  select rs into :displayrs from dual;
  dbms_lob.freetemporary(rs);
end;
/
select :displayrs from dual;
```

The output returns a document that matches the keyword <code>Oracle1</code>, with the SDATA attribute for the document as w?3 and the relevance score of 3:

```
<ctx_result_set><hitlist><hit><score>3</score>
<sdata name="SEC01">w?3&apos;</sdata></hit></hitlist>
<count>1</count></ctx result set>
```

Related Topics

- SET_SECTION_ATTRIBUTE
- SDATA

Use the ${\tt SDATA}$ operator to perform tests on ${\tt SDATA}$ sections and columns, which contain structured data values.

UPDATE SDATA

UPDATE_SDATA is an index API that modifies the specified SDATA values in the index. This API does not store or modify column values in a base table, where the base table column may have been used as an SDATA section.

ADD_SDATA_SECTION

This procedure adds an SDATA section to a section group. By default, all FILTER BY columns are mapped as SDATA.

Oracle Text Application Developer's Guide

8.12 ADD SEC GRP ATTR VAL

Adds a section group attribute value to the list of values of an already existing section group attribute.

Syntax

group_name

Specify the section group name.

attribute name

Specify the name of the section group attribute.

attribute value

Specify the section group attribute value.

8.13 ADD_SPECIAL_SECTION

Adds a special section, either SENTENCE or PARAGRAPH, to a section group. This enables searching within sentences or paragraphs in documents with the WITHIN operator.

A special section in a document is a section which is not explicitly tagged like zone and field sections. The start and end of special sections are detected when the index is created. Oracle Text supports two such sections: *paragraph* and *sentence*.

The sentence and paragraph boundaries are determined by the lexer. For example, the lexer recognizes sentence and paragraph section boundaries as follows:

Table 8-1 Paragraph and Sentence Section Boundaries

Special Section	Boundary
SENTENCE	WORD/PUNCT/WHITESPACE
SENTENCE	WORD/PUNCT/NEWLINE
PARAGRAPH	WORD/PUNCT/NEWLINE/WHITESPACE (indented paragraph)
PARAGRAPH	WORD/PUNCT/NEWLINE/NEWLINE (block paragraph)

The punctuation, whitespace, and newline characters are determined by your lexer settings and can be changed.

If the lexer cannot recognize the boundaries, no sentence or paragraph sections are indexed.

Syntax

group_name

Specify the name of the section group.

section_name

Specify SENTENCE or PARAGRAPH.

Example

The following example enables searching within sentences within HTML documents:

```
begin
ctx_ddl.create_section_group('htmgroup', 'HTML_SECTION_GROUP');
ctx_ddl.add_special_section('htmgroup', 'SENTENCE');
end;
```

Add zone sections to the group to enable zone searching in addition to sentence searching. The following example adds the zone section Headline to the section group htmgroup:

```
ctx ddl.create section group('htmgroup', 'HTML SECTION GROUP');
ctx_ddl.add_special_section('htmgroup', 'SENTENCE');
ctx ddl.add zone section('htmgroup', 'Headline', 'H1');
end:
```

If you are only interested in sentence or paragraph searching within documents and not interested in defining zone or field sections, then use the NULL SECTION GROUP as follows:

```
begin
ctx_ddl.create_section_group('nullgroup', 'NULL_SECTION_GROUP');
ctx_ddl.add_special_section('nullgroup', 'SENTENCE');
end:
Related Topics
```

```
"WITHIN"
"Section Group Types"
"CREATE_SECTION_GROUP"
"ADD ZONE SECTION"
"ADD_FIELD_SECTION"
"REMOVE SECTION"
"DROP_SECTION_GROUP"
```

8.14 ADD STOPCLASS

Adds a stopclass to a stoplist. A stopclass is a class of tokens that is not to be indexed. A stoplist cannot have more than 250 stopclasses with stoppatterns. This does not include the NUMBERS stopclass. When indexing with Stop Patterns, the recommended memory setting is at least 500 MB to 1 GB to optimize the performance of indexing.

English is the only language supported for stopclasses.

Syntax

```
CTX DDL.ADD STOPCLASS (
 stoplist_name IN VARCHAR2,
                       IN VARCHAR2,
IN VARCHAR2 default NULL
 stopclass
stoppattern
```

stoplist_name

Specify the name of the stoplist.

stopclass

Specify the stopclass to be added to stoplist name. It can be either the NUMBERS stopclass or else it is considered as the pattern stopclass.

NUMBERS includes tokens that follow the number pattern: digits, numgroup, and numjoin only. Therefore, 123ABC is not a number, nor is A123. These are labeled as MIXED. \$123 is not a number (this token is not common in a text index because non-alphanumerics become whitespace by default). In the United States, 123.45 is a number, but 123.456.789 is not; in Europe, where numgroup may be '.', the reverse is true.



If NUMBERS is not specified for the stopclass parameter, then it is treated as a pattern stopclass, and you can provide any name to the stopclass parameter. If you specify stopclass as a pattern class, then you need to specify the pattern in the stoppattern parameter. The pattern includes any string pattern that may contain numbers and dates as well. The maximum number of stopwords, stopthemes, and stopclasses you can add to a stoplist is 4095.

stoppattern

Specify the stop pattern to add to the stoplist. If the stopclass is specified as a pattern class, then the stop pattern must be specified. You can use the Oracle Regular Expression to specify the stop pattern.

Call the ADD_STOPCLASS procedure multiple times to add multiple stop patterns to a stoplist. You must specify different stopclass names for adding multiple stop patterns to a stoplist. A stop pattern is not case-sensitive by default, but acts as case-sensitive when the MIXED_CASE lexer preference is enabled. The stop pattern can have the maximum length of 512 characters. When indexing with Stop Patterns, the recommended memory setting is at least 500 MB to 1 GB to optimize the performance of indexing.



Oracle Database Development Guide for more information about the syntax of the Oracle Regular Expression.

Example

The following example adds a stopclass of NUMBERS to the stoplist mystoplist:

```
begin
ctx_ddl.add_stopclass('mystoplist', 'NUMBERS');
end:
```

The following example adds the pattern stopclass of SSN to the stoplist mystoplist:

```
begin
ctx_ddl.add_stopclass('mystoplist', 'SSN', '\d{3}-\d{2}-\d{4}');
end;
```

In this example, the stopclass SSN matches all the tokens of the form <3 digit number>-<2 digit number>-<4 digit number>, example, 234-11-8902.

Related Topics

```
"CREATE_STOPLIST"

"REMOVE_STOPCLASS"

"DROP_STOPLIST"
```

8.15 ADD STOP SECTION

Adds a stop section to an automatic section group. Adding a stop section causes the automatic section indexing operation to ignore the specified section in XML documents.



Adding a stop section causes no section information to be created in the index. However, the text within a stop section is always searchable.

Adding a stop section is useful when your documents contain many low information tags. Adding stop sections also improves indexing performance with the automatic section group.

The number of stop sections you can add is unlimited.

Stop sections do not have section names and hence are not recorded in the section views.

Syntax

section_group

Specify the name of the automatic section group. If you do not specify an automatic section group, then this procedure returns an error.

tag

Specify the tag to ignore during indexing. This parameter is case-sensitive. Defining a stop tag as such also stops the tag's attribute sections, if any.

Qualify the tag with document type in the form (doctype) tag. For example, if you wanted to make the <fluff> tag a stop section only within the mydoc document type, specify (mydoc) fluff for tag.

Example

Defining Stop Sections

The following example adds a stop section identified by the tag <fluff> to the automatic section group myauto:

```
begin
ctx_ddl.add_stop_section('myauto', 'fluff');
end;
```

This example also stops any attribute sections contained within <fluff>. For example, if a document contained:

```
<fluff type="computer">
```

Then the preceding example also stops the attribute section fluff@type.

Doctype Sensitive Stop Sections

The following example creates a stop section for the tag <fluff> only in documents that have a root element of mydoc:

```
begin
ctx_ddl.add_stop_section('myauto', '(mydoc)fluff');
end;
```



Related Topics

```
"ALTER INDEX "
```

"CREATE_SECTION_GROUP"

8.16 ADD_STOPTHEME

Adds a single stoptheme to a stoplist. A stoptheme is a theme that is not to be indexed.

In English, query on indexed themes using the ABOUT operator.

Syntax

stoplist_name

Specify the name of the stoplist.

stoptheme

Specify the stoptheme to be added to <code>stoplist_name</code>. The system normalizes the <code>stoptheme</code> you enter using the knowledge base. If the normalized theme is more than one theme, then the system does not process your stoptheme. For this reason, Oracle recommends that you submit single stopthemes.

The maximum number of stopwords, stopthemes, and stopclasses you can add to a stoplist is 4095.

Example

The following example adds the stoptheme banking to the stoplist mystop:

```
begin
ctx_ddl.add_stoptheme('mystop', 'banking');
end;
```

Related Topics

```
"CREATE_STOPLIST"

"REMOVE_STOPTHEME"

"DROP_STOPLIST"

"ABOUT"
```

8.17 ADD_STOPWORD

Use this procedure to add a single stopword to a stoplist.

To create a list of stopwords, you must call this procedure once for each word.

Syntax

```
CTX_DDL.ADD_STOPWORD(
```



```
stoplist_name IN VARCHAR2,
stopword IN VARCHAR2,
language IN VARCHAR2 default NULL,
language_dependent IN BOOLEAN default TRUE
```

stoplist_name

Specify the name of the stoplist.

stopword

Specify the stopword to be added.

Language-specific stopwords must be unique across the other stopwords specific to the language. For example, it is valid to have a German *die* and an English *die* in the same stoplist.

The maximum number of stopwords, stopthemes, and stopclasses you can add to a stoplist is 4095.

language

Specify the language of stopword when the stoplist you specify with stoplist_name is of type MULTI_STOPLIST. You must specify the globalization support name or abbreviation of an Oracle Text-supported language.

To make a stopword active in multiple languages, specify ALL for this parameter. For example, defining ALL stopwords is useful when you have international documents that contain English fragments that need to be stopped in any language.

An ALL stopword is active in all languages. If you use the multi-lexer, the language-specific lexing of the stopword occurs, just as if it had been added multiple times in multiple specific languages.

Otherwise, specify NULL.

language dependent

Set this parameter to FALSE to indicate that any user-defined string can be specified for the language parameter.

Example

Single Language Stoplist

The following example adds the stopwords because, notwithstanding, nonetheless, and therefore to the stoplist mystop:

```
begin

ctx_ddl.add_stopword('mystop', 'because');
ctx_ddl.add_stopword('mystop', 'notwithstanding');
ctx_ddl.add_stopword('mystop', 'nonetheless');
ctx_ddl.add_stopword('mystop', 'therefore');
end;
```

Multi-Language Stoplist

The following example adds the German word *die* to a multi-language stoplist:

```
begin
ctx_ddl.add_stopword('mystop', 'Die', 'german');
end;
```



Adding An ALL Stopword

The following adds the word the as an ALL stopword to the multi-language stoplist globallist:

```
begin
ctx_ddl.add_stopword('globallist','the','ALL');
end;
```

Notes

- Add stopwords after you create the index with ALTER INDEX.
- Stoplists do not affect string-value SDATA sections, that is, if a stopword is present within an SDATA section, then the token will still be indexed and can be queried using the SDATA operator.

Related Topics

- CREATE STOPLIST
- REMOVE STOPWORD
- DROP_STOPLIST
- ALTER INDEX

Use the ALTER INDEX statement to change or rebuild an existing index, such as Oracle Text index, Oracle Text search index, JSON search index, XML search index, or hybrid vector index.

Supplied Stoplists

By default, the system indexes text using the Oracle Text supplied stoplists that correspond to your database language.

8.18 ADD_SUB_LEXER

Adds a sub-lexer to a multi-lexer preference. A sub-lexer identifies a language in a multi-lexer (multi-language) preference. Use a multi-lexer preference when you want to index more than one language.

Syntax

lexer_name

Specify the name of the multi-lexer preference.

language

Specify the globalization support language name or abbreviation of the sub-lexer. For example, specify JAPANESE or JA for Japanese.

The sub-lexer you specify with sub_lexer is used when the language column has a value case-insensitive equal to the globalization support name of abbreviation of language.



Specify DEFAULT to assign a default sub-lexer to use when the value of the language column in the base table is null, invalid, or unmapped to a sub-lexer. The DEFAULT lexer is also used to parse stopwords.

If a sub-lexer definition for language already exists, then it is replaced by this call.

sub lexer

Specify the name of the sub-lexer to use for this language.

alt value

Optionally specify an alternate value for language.

If you specify DEFAULT for language, then you cannot specify an alt value.

The alt_value is limited to 30 bytes and cannot be a globalization support language name, abbreviation, or DEFAULT.

language dependent

Set this parameter to FALSE to indicate that any user-defined string can be specified for the language parameter. If set to FALSE, then the lexing applied to the search expression will not be dependent on the query language. The FALSE option can only be used when a BASIC SECTION GROUP is in use for the index.

Example

This example shows how to create a multi-language text table and how to set up the multi-lexer to index the table.

Create the multi-language table with a primary key, a text column, and a language column as follows:

```
create table globaldoc (
   doc_id number primary key,
   lang varchar2(3),
   text clob
);
```

Assume that the table holds mostly English documents, with an occasional German or Japanese document. To handle the three languages, you must create three sub-lexers: one for English, one for German, and one for Japanese as follows:

```
ctx_ddl.create_preference('english_lexer', 'basic_lexer');
ctx_ddl.set_attribute('english_lexer', 'index_themes', 'yes');
ctx_ddl.set_attribtue('english_lexer', 'theme_language', 'english');
ctx_ddl.create_preference('german_lexer', 'basic_lexer');
ctx_ddl.set_attribute('german_lexer', 'composite', 'german');
ctx_ddl.set_attribute('german_lexer', 'mixed_case', 'yes');
ctx_ddl.set_attribute('german_lexer', 'alternate_spelling', 'german');
ctx_ddl.create_preference('japanese_lexer', 'japanese_vgram_lexer');
```

Create the multi-lexer preference:

```
ctx ddl.create preference('global lexer', 'multi lexer');
```

Because the stored documents are mostly English, make the English lexer the default:

```
ctx_ddl.add_sub_lexer('global_lexer','default','english_lexer');
```

Add the German and Japanese lexers in their respective languages. Also assume that the language column is expressed in ISO 639-2, so add those as alternative values.

```
ctx_ddl.add_sub_lexer('global_lexer','german','german_lexer','ger');
ctx ddl.add sub_lexer('global_lexer','japanese','japanese_lexer','jpn');
```

Create the index globalx, specifying the multi-lexer preference and the language column in the parameters string as follows:

```
create index globalx on globaldoc(text) indextype is ctxsys.context
parameters ('lexer global lexer language column lang');
```

You can specify a user-defined string for the language paramater as follows:

```
ctx_ddl.add_sub_lexer('global_lexer','mysymbol','german_lexer','my_alt_symbol',
language dependent => FALSE);
```

Restrictions

The following restrictions apply to using CTX DDL.ADD SUB LEXER:

- The invoking user must be the owner of the multi-lexer or CTXSYS.
- The lexer name parameter must name a preference which is a multi-lexer lexer.
- A lexer for default must be defined before the multi-lexer can be used in an index.
- The sub-lexer preference owner must be the same as multi-lexer preference owner.
- The sub-lexer preference must not be a multi-lexer lexer.
- A sub-lexer preference cannot be dropped while it is being used in a multi-lexer preference.
- CTX_DDL.ADD_SUB_LEXER records only a reference. The sub-lexer values are copied at create index time to index value storage.

8.19 ADD_ZONE_SECTION

Creates a zone section and adds the section to an existing section group. This enables zone section searching with the WITHIN operator.

Zone sections are sections delimited by start and end tags. The and tags in HTML, for instance, marks a range of words which are to be rendered in boldface.

Zone sections can be nested within one another, can overlap, and can occur more than once in a document.

Syntax

```
CTX_DDL.ADD_ZONE_SECTION(
group_name IN VARCHAR2,
section_name IN VARCHAR2,
tag IN VARCHAR2
);
```

group_name

Specify the name of the section group to which section name is added.

section name

Specify the name of the section to add to the group_name. Use this name to identify the section in WITHIN queries. Avoid using names that contain non-alphanumeric characters such as _, because most of these characters are special must be escaped in queries. Section names are case-insensitive.



Within the same group, zone section names and field section names cannot be the same. The terms *Paragraph* and *Sentence* are reserved for special sections.

Section names need not be unique across tags. You can assign the same section name to more than one tag, making details transparent to searches.

tag

Specify the pattern which marks the start of a section. For example, if $< \pm 1 >$ is the HTML tag, specify $\pm 1 <$ for $\pm ag$. The start tag you specify must be unique within a section group. Oracle Text knows what the end tags look like from the $group_type$ parameter you specify when you create the section group.

If group_name is an HTML_SECTION_GROUP, you can create zone sections for the META tag's NAME/CONTENT attribute pairs. To do so, specify tag as meta@namevalue where namevalue is the value of the NAME attribute whose CONTENT attributes are to be indexed as a section. Refer to the example.

If group_name is an XML_SECTION_GROUP, you can optionally qualify tag with a document type (root element) in the form (doctype) tag. Doing so makes section_name sensitive to the XML document type declaration. Refer to the example.

Examples

Creating HTML Sections

The following example defines a section group called htmgroup of type HTML_SECTION_GROUP. It then creates a zone section in htmgroup called headline identified by the <H1> tag:

```
begin
ctx_ddl.create_section_group('htmgroup', 'HTML_SECTION_GROUP');
ctx_ddl.add_zone_section('htmgroup', 'heading', 'H1');
end:
```

After indexing with section group htmgroup, query within the heading section by issuing a query as follows:

```
'Oracle WITHIN heading'
```

Creating Sections for <META NAME > Tags

You can create zone sections for HTML META tags when you use the HTML SECTION GROUP.

Consider an HTML document that has a META tag as follows:

```
<META NAME="author" CONTENT="ken">
```

To create a zone section that indexes all CONTENT attributes for the META tag whose NAME value is author:

```
begin
ctx_ddl.create_section_group('htmgroup', 'HTML_SECTION_GROUP');
ctx_ddl.add_zone_section('htmgroup', 'author', 'meta@author');
end
```

After indexing with section group htmgroup, query the document as follows:

```
'ken WITHIN author'
```

Creating Document Type Sensitive Sections (XML Documents Only)

You have an XML document set that contains the <book> tag declared for different document types (DTDs). You want to create a distinct book section for each document type.

Assume that myDTDname is declared as an XML document type as follows:

```
<!DOCTYPE myDTDname>
<myDTDname>
...
```

(Note: the DOCTYPE must match the top-level tag.)

Within myDTDname, the element <book> is declared. For this tag, create a section named mybooksec that is sensitive to the tag's document type as follows:

```
begin
ctx_ddl.create_section_group('myxmlgroup', 'XML_SECTION_GROUP');
ctx_ddl.add_zone_section('myxmlgroup', 'mybooksec', '(myDTDname)book');
end;
```

Notes

Repeated Sections

Zone sections can repeat. Each occurrence is treated as a separate section. For example, if <H1> denotes a heading section, they can repeat in the same documents as follows:

```
<H1> The Brown Fox </H1>
```

```
<H1> The Gray Wolf </H1>
```

Assuming that these zone sections are named Heading, the query *Brown WITHIN Heading* returns this document. However, a query of (*Brown and Gray*) *WITHIN Heading* does not.

Overlapping Sections

Zone sections can overlap each other. For example, if and <I> denote two different zone sections, they can overlap in document as follows:

```
plain <B> bold <I> bold and italic </B> only italic </I> plain
```

Nested Sections

Zone sections can nest, including themselves as follows:

```
<TD> <TABLE><TD>nested cell</TD></TABLE></TD>
```

Using the WITHIN operator, you can write queries to search for text in sections within sections. For example, assume the BOOK1, BOOK2, and AUTHOR zone sections occur as follows in documents doc1 and doc2:

doc1:

```
<bookl> <author>Scott Tiger</author> This is a cool book to read.</bookl>
```

doc2:

<book2> <author>Scott Tiger</author> This is a great book to read.

Consider the nested query:

```
'(Scott within author) within book1'
```

This query returns only doc1.



Related Topics

```
"WITHIN"

"Section Group Types"

"CREATE_SECTION_GROUP"

"ADD_FIELD_SECTION"

"ADD_SPECIAL_SECTION"

"REMOVE_SECTION"

"DROP_SECTION_GROUP"
```

8.20 COPY POLICY

Creates a new policy from an existing policy or index.

Syntax

source_policy

The name of the policy or index being copied.

policy_name

The name of the new policy copy.

The preference values are copied from the <code>source_policy</code>. Both the source policy or index and the new policy must be owned by the same database user.

8.21 CREATE_INDEX_SET

Creates an index set for CTXCAT index types.

Name this index set in the parameter clause of CREATE INDEX when you create a CTXCAT index.



The Oracle Text indextype CTXCAT is deprecated with Oracle Database 23ai. The indextype itself, and it's operator CTXCAT, can be removed in a future release. CTXCAT was introduced when indexes were typically a few megabytes in size. Modern, large indexes, can be difficult to manage with CTXCAT. The addition of index sets to CTXCAT can be achieved more effectively by the use of FILTER BY and ORDER BY columns, or SDATA, or both, in the CONTEXT indextype. CTXCAT is therefore rarely an appropriate choice. Oracle recommends that you choose the more efficient CONTEXT indextype.

Syntax

```
CTX_DDL.CREATE_INDEX_SET(set_name in varchar2);
```



set name

Specify the name of the index set. Name this index set in the parameter clause of CREATE INDEX when you create a CTXCAT index.

8.22 CREATE_PATH_LIST

Creates a path-list preference to use with the PATHLIST parameter, when creating a path-subsetting JSON search index.

Understand Path Subsetting

You can use **path subsetting** to identify the fields in a document that you want to include or exclude from indexing. The excluded fields are not indexed, and the JSON search index is not used for them when querying. Filtering out irrelevant paths from documents can reduce the amount of data indexed, thereby minimizing disk space, indexing costs, and the index creation or rebuild time. As a result, the index maintenance operations improve in performance by skipping irrelevant data and saving further processing.

Consider using path subsetting if your documents include:

- · A subset of paths that is frequently queried or is relevant for indexing
- Infrequently queried data or an irrelevant set of paths that can be ignored during indexing

As an alternative to specifying the INCLUDE or EXCLUDE clauses (as explained in CREATE SEARCH INDEX for JSON search index SEARCH_ON options), you can also use the PL/SQL subprograms CTX_DDL.CREATE_PATH_LIST and CTX_DDL.ADD_PATH to specify a subset of paths to include or exclude from indexing. You can then use the CTX_DDL.ADD_PATH API to add paths to this pathlist.

Syntax

```
begin
  CTX_DDL.CREATE_PATH_LIST(
    pref_name IN VARCHAR2,
    format IN NUMBER,
    behavior IN VARCHAR2
);
end;
```

pref name

Specify the name of the pathlist preference, which is a list of paths to include or exclude from indexing.

format

Specify the type of document that this pathlist should support. Currently, the only supported value is CTX DDL.PATHLIST JSON.



You can enable path-subsetting for JSON search indexes only if the indexed column's data-type is JSON type.

behavior

Specify the behavior of the pathlist. The values can be:

- CTX DDL.PATHLIST INCLUDE to create a list of path to be included for indexing
- CTX DDL.PATHLIST EXCLUDE to create a list of path to be excluded from indexing

This argument is optional. When omitted, the behavior value defaults to CTX_DDL.PATHLIST_INCLUDE.

Restrictions

- Specifying any SEARCH_ON clause while there is also a PATHLIST parameter results in an
 error. Similarly, you cannot specify a PATHLIST parameter for an index that has the
 Dataguide feature enabled.
- You can specify a path subsetting clause with SEARCH_ON TEXT, TEXT_VALUE, and VALUE (not with NONE and TEXT VALUE STRING).
- You cannot specify both the INCLUDE and EXCLUDE clauses for a single index.

Examples

To create an INCLUDE pathlist:

The following statement creates a JSON search index with *path subsetting* for full-text and string-equality searches of fields \$.SpecialInstructions and \$.LineItems.Part.Description of a purchase order document. But it also indexes

fields \$.PoNumber and \$.LineItems.Part.UnitPrice for numeric-value ranges, and fields \$.Reference, \$.User, \$.ShippingInstructions.name, and \$.ShippingInstructions.Address.zipCode for string-value ranges.

and 3. shipping instructions. Address. 21pcode for string-value ranges.

```
CREATE SEARCH INDEX json_idx ON json_tab (purchase_order_jsondoc)

FOR JSON PARAMETERS ('SEARCH_ON

TEXT INCLUDE ($.SpecialInstructions, $.LineItems.Part.Description)

VALUE(NUMBER) INCLUDE ($.PONumber, $.LineItems.Part.UnitPrice)

VALUE(VARCHAR2) INCLUDE ($.Reference,

$.User,

$.ShippingInstructions.name,

$.ShippingInstructions.Address.zipCode)');
```

Alternatively, you can create the same index using these pathlist APIs.

Here, you first create a list of paths (json_pl_incl) to be included for indexing using the CTX_DDL.CREATE_PATH_LIST and CTX_DDL.ADD_PATH APIs. You then create an index using the PATHLIST parameter, whose value is a named list of the paths to be included.

```
-- create a pathlist preference and add paths

BEGIN

CTX_DDL.create_path_list('json_pl_incl', CTX_DDL.PATHLIST_JSON,

CTX_DDL.PATHLIST_INCLUDE);

CTX_DDL.add_path('json_pl', 'TEXT', '$.SpecialInstructions');

CTX_DDL.add_path('json_pl', 'TEXT', '$.LineItems.Part.Description');

CTX_DDL.add_path('json_pl', 'NUMBER', '$.PONumber');

CTX_DDL.add_path('json_pl', 'NUMBER', '$.LineItems.Part.UnitPrice');

CTX_DDL.add_path('json_pl', 'VARCHAR2', '$.Reference');

CTX_DDL.add_path('json_pl', 'VARCHAR2', '$.User');
```



```
CTX_DDL.add_path('json_pl', 'VARCHAR2', '$.ShippingInstructions.name');
CTX_DDL.add_path('json_pl', 'VARCHAR2',
'$.ShippingInstructions.Address.zipCode');
END;

-- declare the pathlist preference name

CREATE SEARCH INDEX json_idx ON json_tab (purchase_order_jsondoc)
    FOR JSON PARAMETERS ('PATHLIST json pl incl');
```

To create an EXCLUDE pathlist:

Here, you first create a list of paths (json_pl_excl) to be excluded from indexing using the CTX_DDL.CREATE_PATH_LIST and CTX_DDL.ADD_PATH APIs. You then create an index using the PATHLIST parameter, whose value is a named list of the paths to be excluded.

```
-- create a pathlist preference and add paths
BEGIN
  CTX DDL.create path list('json pl excl', CTX DDL.PATHLIST JSON,
CTX DDL.PATHLIST EXCLUDE);
  CTX DDL.add path('json pl excl', 'TEXT',
                                                '$.SpecialInstructions');
  CTX DDL.add path('json pl excl', 'TEXT',
'$.LineItems.Part.Description');
  CTX DDL.add path('json pl excl', 'NUMBER',
                                               '$.PONumber');
  CTX DDL.add path('json pl excl', 'NUMBER',
'$.LineItems.Part.UnitPrice');
  CTX_DDL.add_path('json_pl excl', 'VARCHAR2', '$.Reference');
  CTX DDL.add path('json pl excl', 'VARCHAR2', '$.User');
 CTX DDL.add path('json pl excl', 'VARCHAR2',
'$.ShippingInstructions.name');
 CTX DDL.add_path('json pl excl', 'VARCHAR2',
'$.ShippingInstructions.Address.zipCode');
END;
-- declare the pathlist preference name
CREATE SEARCH INDEX json idx ON json tab (purchase order jsondoc)
  FOR JSON PARAMETERS ('PATHLIST json pl excl');
```

Related Topics

CREATE SEARCH INDEX

Use the <code>CREATE SEARCH INDEX</code> statement to create a search index for indexing and querying structured, unstructured, or semi-structured data, such as textual, JSON, and XML documents.

ADD_PATH

Adds a path to a path-list preference for use with the PATHLIST parameter, when creating a path-subsetting JSON search index.

DROP_PATH_LIST

Removes an existing pathlist preference.

8.23 CREATE POLICY

Creates a policy to use with the CTX_DOC.POLICY_* procedures, certain Oracle Data Mining procedures, and the in-memory Text index.

Syntax

policy_name

Specify the name for the new policy. Policy names and Text indexes share the same namespace.

filter

Specify the filter preference to use.

section group

Specify the section group to use. You can specify any section group that is supported by CONTEXT index.

lexer

Specify the lexer preference to use. Your INDEX THEMES attribute must be disabled.

stoplist

Specify the stoplist preference to use.

wordlist

Specify the wordlist preference to use.

datastore

Specify the datastore preference to use for the in-memory Text index.



The datastore parameter is only supported for the in-memory Text index.

Examples

Create a lexer preference named mylex.

```
begin
    ctx_ddl.create_preference('mylex', 'BASIC_LEXER');
    ctx_ddl.set_attribute('mylex', 'printjoins', '_-');
end:
```

Create a stoplist preference named mystop.

```
begin
   ctx ddl.create stoplist('mystop', 'BASIC STOPLIST');
   ctx_ddl.add_stopword('mystop', 'because');
   ctx ddl.add stopword('mystop', 'nonetheless');
   ctx_ddl.add_stopword('mystop', 'therefore');
end;
Create a wordlist preference named mywordlist.
begin
   ctx ddl.create preference('mywordlist', 'BASIC WORDLIST');
Create a datastore preference named my file datastore.
begin
   create or replace directory wdirectory as '/path1/path2';
   GRANT read ON DIRECTORY WDIRECTORY TO user;
   EXEC ctx ddl.create preference('my file datastore', 'DIRECTORY DATASTORE');
   EXEC ctx ddl.set attribute('my file datastore', 'DIRECTORY', 'WDIRECTORY');
end;
Create a policy named mypolicy.
exec ctx ddl.create policy(
   'mypolicy',
   NULL,
   NULL,
   'mylex',
   'mystop',
   'mywordlist',
   'my file datastore'
);
or
exec ctx ddl.create policy(
  policy name => 'mypolicy',
  lexer => 'mylex',
  stoplist => 'mystop',
  wordlist => 'mywordlist',
   datastore => 'my_file_datastore'
);
Use ALTER TABLE to apply your defined policy to a column for in-memory search. Then enter
your query statement, using the CONTAINS operator in the WHERE clause:
ALTER TABLE my_tab INMEMORY TEXT(my_txt_col using 'mypolicy');
SELECT id from my_tab WHERE CONTAINS(my_txt_col, 'Washington')>0;
Update the policy with the following:
exec ctx ddl.update policy(
   policy_name => 'mypolicy',
   lexer => 'my new lex'
);
Drop the policy with the following:
exec ctx_ddl.drop_policy(policy_name => 'mypolicy');
```



Related Topics

Oracle Text Application Developer's Guide

8.24 CREATE PREFERENCE

Creates a preference in the Text data dictionary.

Specify preferences in the parameter string of CREATE INDEX or ALTER INDEX.



Caution:

CTX_DDL.CREATE_PREFERENCE does not respect the current schema as set by ALTER SESSION SET current_schema. Therefore, if you need to create or delete a preference owned by another user, then you must explicitly state this, and you must have the CREATE ANY TABLE system privilege.

Syntax

```
CTX_DDL.CREATE_PREFERENCE(preference_name in varchar2, object name in varchar2);
```

preference name

Specify the name of the preference to be created.

object_name

Specify the name of the preference type.



See Also:

For a complete list of preference types and their associated attributes, see Oracle Text Indexing Elements

Examples

Creating Text-only Index

The following example creates a lexer preference that specifies a text-only index. It does so by creating a BASIC_LEXER preference called my_lexer with CTX_DDL.CREATE_PREFERENCE. It then calls CTX_DDL.SET_ATTRIBUTE twice, first specifying YES for the INDEX_TEXT attribute, then specifying NO for the INDEX_THEMES attribute.

```
ctx_ddl.create_preference('my_lexer', 'BASIC_LEXER');
ctx_ddl.set_attribute('my_lexer', 'INDEX_TEXT', 'YES');
ctx_ddl.set_attribute('my_lexer', 'INDEX_THEMES', 'NO');
end;
```

Specifying File Data Storage

The following example creates a data storage preference called mypref that tells the system that the files to be indexed are stored in an Oracle directory object. The example then uses CTX_DDL.SET_ATTRIBUTE to set the DIRECTORY attribute to the directory /docs.

```
begin
ctx_ddl.create_preference('mypref', 'DIRECTORY_DATASTORE');
ctx_ddl.set_attribute('mypref', 'DIRECTORY', '/docs');
end:
```

See Also:

For more information about data storage, see "Datastore Types"

Creating Primary/Detail Relationship

Use CTX_DDL.CREATE_PREFERENCE to create a preference with <code>DETAIL_DATASTORE</code>. Use CTX_DDL.SET_ATTRIBUTE to set the attributes for this preference. The following example shows how this is done:

```
begin
ctx_ddl.create_preference('my_detail_pref', 'DETAIL_DATASTORE');
ctx_ddl.set_attribute('my_detail_pref', 'binary', 'true');
ctx_ddl.set_attribute('my_detail_pref', 'detail_table', 'my_detail');
ctx_ddl.set_attribute('my_detail_pref', 'detail_key', 'article_id');
ctx_ddl.set_attribute('my_detail_pref', 'detail_lineno', 'seq');
ctx_ddl.set_attribute('my_detail_pref', 'detail_text', 'text');
end;
```

See Also:

For more information about primary/detail, see "DETAIL DATASTORE"

Specifying Storage Attributes

The following examples specify that the index tables are to be created in the foo tablespace with an initial extent of 1K:





If S_{TABLE_CLAUSE} is specified for a storage preference in an index without SDATA, then it has no effect on the index, and the index creation will still succeed.

See Also:

Storage Types

Creating Preferences with No Attributes

When you create preferences with types that have no attributes, you need only create the preference, as in the following example which sets the filter to the NULL FILTER:

```
begin
ctx_ddl.create_preference('my_null_filter', 'NULL_FILTER');
end;
```

Specifying BIGRAM Mode for Japanese VGRAM Lexer

The following example creates a Japanese VGRAM lexer preference that specifies the BIGRAM mode of operation for the Japanese queries:

```
begin
ctx_ddl.create_preference('jp_lexer','JAPANESE_VGRAM_LEXER');
ctx_ddl.set_attribute('jp_lexer','BIGRAM','TRUE');
end;

/* create the index */
create index jp_idx on jp_doc(text) indextype is ctxsys.context
    parameters('lexer jp lexer');
```

Related Topics

SET ATTRIBUTE

DROP_PREFERENCE

CREATE INDEX

ALTER INDEX

Oracle Text Indexing Elements

8.25 CREATE_SECTION_GROUP

Creates a section group for defining sections in a text column.

When you create a section group, you can add to it zone, field, or special sections with ADD_ZONE_SECTION, ADD_FIELD_SECTION, ADD_MDATA_SECTION, or ADD_SPECIAL_SECTION.

You also use <code>CREATE_SECTION_GROUP</code> with <code>CTX_DDL.SET_SEC_GRP_ATTR</code> to set <code>xml_enable</code> to create an Oracle XML Search Index.

When you index, name the section group in the parameter string of CREATE INDEX or ALTER INDEX .

After indexing, query within your defined sections with the WITHIN operator.

Syntax

```
CTX_DDL.CREATE_SECTION_GROUP(
  group_name in varchar2,
  group_type in varchar2
);
```

group_name

Specify the section group name to create as <code>section_group_name</code>. This parameter must be unique within an owner.

group_type

Specify section group type. The group type parameter can be one of the following:

Section Group Preference	Description
NULL_SECTION_GROUP	Use this group type when you define no sections or when you define <i>only</i> SENTENCE or PARAGRAPH sections. This is the default.
BASIC_SECTION_GROUP	Use this group type for defining sections where the start and end tags are of the form <a> and . Note: This group type does not support input such as unbalanced parentheses, comments tags, and attributes. Use HTML_SECTION_GROUP for this type of input.
HTML_SECTION_GROUP	Use this group type for indexing HTML documents and for defining sections in HTML documents.
JSON_SECTION_GROUP	Use this group to create a JSON enabled context index. The JSON ENABLE attribute cannot be used with XML ENABLE. A section group can only be marked as JSON ENABLE. If it is already marked with XML ENABLE, then the path section group cannot be used for JSON ENABLE and vice versa.
XML_SECTION_GROUP	Use this group type for indexing XML documents and for defining sections in XML documents.



Section Group Preference	Description
AUTO_SECTION_GROUP	Use this group type to automatically create a zone section for each start-tag/end-tag pair in an XML document. The section names derived from XML tags are case sensitive as in XML. Attribute sections are created automatically for XML tags that have attributes. Attribute sections are named in the form attribute@tag. Stop sections, empty tags, processing instructions, and comments are not indexed. The following limitations apply to automatic section groups: You cannot add zone, field, or special sections to an automatic section group. Automatic sectioning does not index XML document types (root elements.) However, you can define stop sections with document type. The length of the indexed tags, including prefix and namespace, cannot exceed 64 bytes. Tags longer than this are not indexed.
PATH_SECTION_GROUP	Use this group type to index XML documents. Behaves like the AUTO_SECTION_GROUP. The difference is that with this section group you can do path searching with the INPATH and HASPATH operators. Queries are also case-sensitive for tag and attribute names.
NEWS_SECTION_GROUP	Use this group for defining sections in newsgroup formatted documents according to RFC 1036.

Note:

Starting with Oracle Database 18c, use of <code>NEWS_SECTION_GROUP</code> is deprecated in Oracle Text. Use external processing instead.

If you want to index USENET posts, then preprocess the posts to use ${\tt BASIC_SECTION_GROUP} \ or \ {\tt HTML_SECTION_GROUP} \ within \ Oracle \ Text. \ USENET \ is \ rarely \ used \ commercially.$

Examples

The following command creates a section group called htmgroup with the HTML group type.

```
begin
ctx_ddl.create_section_group('htmgroup', 'HTML_SECTION_GROUP');
end;
```

The following command creates a section group called auto with the AUTO_SECTION_GROUP group type to be used to automatically index tags in XML documents.

```
begin
ctx_ddl.create_section_group('auto', 'AUTO_SECTION_GROUP');
```

end:

The following example creates an Oracle XML Search index:

```
exec CTX DDL.CREATE SECTION GROUP('secgroup', 'PATH SECTION GROUP');
exec CTX DDL.SET SEC GRP ATTR('secgroup','xml enable','t');
CREATE INDEX po ctx idx on T(X) indextype is ctxsys.context
parameters ('section group SECGROUP');
Related Topics
```

```
"WITHIN"
"Section Group Types"
"ADD_ZONE_SECTION"
"ADD FIELD SECTION"
"ADD_MDATA_SECTION"
"ADD_SPECIAL_SECTION "
"REMOVE SECTION"
"DROP SECTION GROUP"
```

8.26 CREATE SHADOW INDEX

Creates index metadata (or policy) for the specified index. If the index is not partitioned, then it also creates the index tables. This procedure is only supported in Enterprise Edition of Oracle Database.

The following changes are not supported:

- Transition from non-composite domain index to composite, or changing the composite domain index columns.
- Rebuild indexes that have partitioned index tables, for example, \$1, \$P, \$K.

Note:

- For a partitioned index, you must first call this procedure to create the shadow index metadata. This procedure will not create index tables. It has no effect on guery, DML, sync, or optimize operations.
- The CREATE SHADOW INDEX and RECREATE INDEX ONLINE procedures do not support section group with the XML ENABLE attribute on CONTEXT indexes. Doing so results in the "DRG-10521: Operation not supported with XML ENABLE on a CONTEXT Index" error.

Syntax

```
CTX DDL.CREATE SHADOW INDEX(
 idx name IN VARCHAR2,
 parameter string IN VARCHAR2 DEFAULT NULL,
 parallel degree IN NUMBER DEFAULT 1
);
```



idx name

The name of a valid CONTEXT indextype.

parameter_string

For nonpartitioned index, the same string as in ALTER INDEX. For partitioned index, the same string as in ALTER INDEX PARAMETER.

parallel_degree

Reserved for future use. Specify the degree of parallelism. Parallel operation is not currently supported.

Example

Example 8-1 Scheduled Global Index RECREATE (Incremental Rebuild)

In this example, you have the finest control over each stage of RECREATE_INDEX_ONLINE. Since <code>SYNC_INDEX</code> can take a time limit, you can limit <code>SYNC_INDEX</code> during non-business hours and incrementally recreate the index.

```
/* create lexer and original index */
exec ctx ddl.create preference('us lexer', 'basic lexer');
create index idx on tbl(text) indextype is ctxsys.context
  parameters('lexer us lexer');
/* create a new lexer */
begin
  ctx_ddl.create_preference('e_lexer','basic_lexer');
  ctx_ddl.set_attribute('e_lexer', 'base_letter', 'yes');
  ctx_ddl.create_preference('m_lexer', 'multi_lexer');
  ctx ddl.add sub lexer('m lexer', 'default', 'us lexer');
  ctx_ddl.add_sub_lexer('m_lexer','e','e_lexer');
end;
/* add new language column to the table for multi-lexer */
alter table tbl add(lang varchar2(10) default 'us');
/* create shadow index */
exec ctx ddl.create shadow index('idx',
  'replace lexer m lexer language column lang NOPOPULATE');
declare
  idxid integer;
begin
  /* figure out shadow index name */
  select idx id into idxid from ctx user indexes
    where idx name = 'IDX';
  /* populate pending */
  ctx_ddl.populate_pending('RIO$'||idxid);
  /* time limited sync */
  ctx_ddl.sync_index(idx_name =>'RIO$'||idxid,
                    maxtime =>480);
  /* more sync until no pending rows for the shadow index */
end;
/* swap in the shadow index */
exec ctx ddl.exchange shadow index('idx');
```

Notes

• The index name for the shadow index is RIO\$index_id. By default, it also populates index tables for nonpartitioned indexes, unless NOPOPULATE is specified in CREATE INDEX or in

ALTER INDEX. For a local partitioned index, it only creates index metadata without creating the index tables for each partition. Each index can have only one shadow index.

• When building a nonpartitioned index online, you can first call this procedure to create index metadata and index tables. If you specify POPULATE, then this procedure populates the index, but does not do swapping. You can schedule the swapping at a later, preferred time.

If you specify NOPOPULATE, it only creates metadata for the index tables, but does not populate them. You must perform POPULATE_PENDING (CTX_DDL.POPULATE_PENDING) to populate the pending queues after running this procedure, and then sync the indexes. This is referred to as *incremental re-create*.

Queries are all processed normally when this procedure is running.

- If POPULATE is specified, then DML is blocked for a very short time at the beginning of populate, after which all further DML is logged into an online pending queue and processed later.
- In case of NOPOPULATE shadow indexes, ensure that you execute the POPULATE_PENDING procedure before calling a DML operation. If you call a DML operation before executing the POPULATE PENDING procedure, then the same tokens appear twice in the \$1 index table.
- Sync with CTX_DDL.SYNC_INDEX runs normally on the index. OPTIMIZE_INDEX runs without doing anything, but does not return an error.
- When you change the SYNC type for a shadow index, the EXCHANGE_SHADOW_INDEX procedure swaps the main index with the new SYNC type. However, the shadow index continues to use MANUAL synchronization as the SYNC type. This feature enables you to control when you want to populate or exchange the shadow index.

Related Topics

ALTER INDEX

Use the ALTER INDEX statement to change or rebuild an existing index, such as Oracle Text index, Oracle Text search index, JSON search index, XML search index, or hybrid vector index.

CREATE INDEX

Use the CREATE INDEX statement to create an Oracle Text index.

Oracle Text SQL Statements and Operators

These are the SQL statements and Oracle Text operators for creating and managing Oracle Text indexes and performing Oracle Text queries.

- DROP_SHADOW_INDEX
- EXCHANGE SHADOW INDEX

This procedure swaps the index (or index partition) metadata and index (or index partition) data.

SYNC INDEX

Synchronizes the index to process inserts, updates, and deletes to the base table.

POPULATE PENDING

This procedure populates the pending queue with every ROWID in the base table or table partition. This procedure is only supported for CONTEXT indexes.

8.27 CREATE_STOPLIST

Use this procedure to create a new, empty stoplist. Stoplists can contain words or themes that are not to be indexed.

You can also create multi-language stoplists to hold language-specific stopwords. A multi-language stoplist is useful when you index a table that contains documents in different languages, such as English, German, and Japanese. When you do so, the text table must contain a language column.

Add either stopwords, stopclasses, or stopthemes to a stoplist using ADD_STOPWORD, ADD_STOPCLASS, or ADD_STOPTHEME. Specify a stoplist in the parameter string of CREATE INDEX or ALTER INDEX to override the default stoplist CTXSYS.DEFAULT STOPLIST.

Syntax

```
CTX_DDL.CREATE_STOPLIST(
stoplist_name IN VARCHAR2,
stoplist_type IN VARCHAR2 DEFAULT 'BASIC_STOPLIST');
```

stoplist_name

Specify the name of the stoplist to be created.

stoplist_type

Specify <code>BASIC_STOPLIST</code> to create a stoplist for a single language. This is the default. Specify <code>MULTI_STOPLIST</code> to create a stoplist with language-specific stopwords. At indexing time, the language column of each document is examined, and only the stopwords for that language are eliminated. At query time, the session language setting determines the active stopwords, like it determines the active lexer when using the multi-lexer.



When indexing a multi-language table with a multi-language stoplist, the table must have a language column.

Examples

Example 8-2 Single Language Stoplist

The following example creates a stoplist called mystop:

```
begin
ctx_ddl.create_stoplist('mystop', 'BASIC_STOPLIST');
end;
```

Example 8-3 Multi-Language Stoplist

The following example creates a multi-language stoplist called multistop and then adds tow language-specific stopwords:

```
begin
ctx_ddl.create_stoplist('multistop', 'MULTI_STOPLIST');
ctx_ddl.add_stopword('mystop', 'Die', 'german');
ctx_ddl.add_stopword('mystop', 'Or', 'english');
end;
```

Related Topics

```
"ADD_STOPWORD "
"ADD_STOPCLASS "
"ADD_STOPTHEME "
```



```
"DROP_STOPLIST"
"CREATE INDEX"
"ALTER INDEX"
Supplied Stoplists
```

8.28 DROP_INDEX_SET

Drops a CTXCAT index set created with CTX DDL.CREATE INDEX SET.



The Oracle Text indextype CTXCAT is deprecated with Oracle Database 23ai. The indextype itself, and it's operator CTXCAT, can be removed in a future release. CTXCAT was introduced when indexes were typically a few megabytes in size. Modern, large indexes, can be difficult to manage with CTXCAT. The addition of index sets to CTXCAT can be achieved more effectively by the use of FILTER BY and ORDER BY columns, or SDATA, or both, in the CONTEXT indextype. CTXCAT is therefore rarely an appropriate choice. Oracle recommends that you choose the more efficient CONTEXT indextype.

Syntax

set name

Specify the name of the index set to drop. Dropping an index set drops all of the sub-indexes it contains.

Related Topics

• CREATE_INDEX_SET
Creates an index set for CTXCAT index types.

8.29 DROP_PATH_LIST

Removes an existing pathlist preference.

Syntax

pref name

Specify the name of the pathlist preference to drop.

Example

```
begin
  CTX_DDL.DROP_PATH_LIST(
    'json_pl',
);
end;
```

Related Topics

CREATE_PATH_LIST

Creates a path-list preference to use with the PATHLIST parameter, when creating a path-subsetting JSON search index.

8.30 DROP_POLICY

Drops a policy created with CTX_DDL.CREATE_POLICY.

Syntax

policy_name

Specify the name of the policy to drop.

8.31 DROP_PREFERENCE

The DROP_PREFERENCE procedure deletes the specified preference from the Text data dictionary. Dropping a preference does not affect indexes that have already been created using that preference.

Syntax

preference_name

Specify the name of the preference to be dropped.

Example

The following example drops the preference my lexer.

```
begin
ctx_ddl.drop_preference('my_lexer');
end:
```

Related Topics

CTX_DDL.CREATE_PREFERENCE



8.32 DROP_SECTION_GROUP

The DROP_SECTION_GROUP procedure deletes the specified section group, as well as all the sections in the group, from the Text data dictionary.

Syntax

group_name

Specify the name of the section group to delete.

Example

The following example drops the section group htmgroup and all its sections:

```
begin
ctx_ddl.drop_section_group('htmgroup');
end;
```

Related Topics

```
"CREATE_SECTION_GROUP"

"PREFERENCE_IMPLICIT_COMMIT"
```

8.33 DROP_SHADOW_INDEX

Drops a shadow index for the specified index. When you drop a shadow index, if it is partitioned, then its metadata and the metadata of all this shadow index's partitions are dropped. This procedure also drops all the shadow index tables and cleans up any online pending queue.

Syntax

```
CTX_DDL.DROP_SHADOW_INDEX(
    idx_name in VARCHAR2
);
```

idx name

The name of a valid CONTEXT indextype.

Example

The following example drops the shadow index myshadowidx:

```
begin
ctx_ddl.drop_shadow_index('myshadowidx');
end;
```

Related Topics

CTX_DDL.CREATE_SHADOW_INDEX



8.34 DROP_STOPLIST

Drops a stoplist from the Text data dictionary. When you drop a stoplist, you must re-create or rebuild the index for the change to take effect.

Syntax

```
CTX DDL.DROP STOPLIST(stoplist name in varchar2);
```

stoplist_name

Specify the name of the stoplist.

Example

The following example drops the stoplist mystop:

```
begin
ctx_ddl.drop_stoplist('mystop');
end;
```

Related Topics

CTX_DDL.CREATE_STOPLIST

8.35 EXCHANGE_SHADOW_INDEX

This procedure swaps the index (or index partition) metadata and index (or index partition) data.

For nonpartitioned indexes, this procedure swaps both the metadata and the index data, and processes the online pending queue.

Syntax

idx_name

Specify the name of the CONTEXT indextype.

partition_name

Specify the name of the shadow index partition. May also be NULL.

Example

Example 8-4 Global Index RECREATE with Scheduled Swap

This example demonstrates running CTX_DDL.EXCHANGE_SHADOW_INDEX during non-business hours when query failures and DML blocking can be tolerated.

```
/* create lexer and original index */
exec ctx_ddl.create_preference('us_lexer','basic_lexer');
create index idx on tbl(text) indextype is ctxsys.context
  parameters('lexer us_lexer');
/* create a new lexer */
begin
```



```
ctx_ddl.create_preference('e_lexer','basic_lexer');
  ctx_ddl.set_attribute('e_lexer','base_letter','yes');
  ctx_ddl.create_preference('m_lexer','multi_lexer');
  ctx_ddl.add_sub_lexer('m_lexer','default','us_lexer');
  ctx_ddl.add_sub_lexer('m_lexer','e','e_lexer');
end;
/* add new language column to the table for multi-lexer */
alter table tbl add(lang varchar2(10) default 'us');
/* recreate index online with the new multip-lexer */
exec ctx_ddl.create_shadow_index('idx',
    'replace lexer m_lexer language column lang');
exec ctx ddl.exchange shadow index('idx');
```

Notes

Using EXCHANGE_SHADOW_INDEX with Nonpartitioned Indexes:

For nonpartitioned indexes, this procedure will swap both metadata and index data, and will process the online pending queue.

Queries will return *column not indexed* errors when swapping metadata and index data, but queries are processed normally when processing online pending queue. The period of errors being raised should be short.

If you specify POPULATE when you create the shadow index, and if many DML operations have been issued since the creation of the shadow index, then there could be a large pending queue. However, if you use *incremental recreate*, that is, specify NOPOPULATE when you create the shadow index, and you then populate the pending queue and sync, then the online pending queue is always empty no matter how many DML operations have occurred since CREATE SHADOW INDEX was issued.

When this procedure is running, DML will first fail with an error about index being in inprogress status. After that DML could be blocked if there are rows in online pending queue that need to be reapplied.

Note:

When this procedure is running, DML statements will fail with an error that the index is in "in-progress status." If, when this error occurs, there are rows in the online pending queue that need to be reapplied, then the DML could be blocked and stop responding.

Using EXCHANGE_SHADOW_INDEX with Partitioned Indexes:

For partitions that are recreated with NOSWAP: when the index is partitioned, and if partition_name is a valid index partition, then this procedure will swap the index partition data and the index partition metadata, and will process the online pending queue for this partition.

This procedure swaps only one partition at a time. When you run this procedure on partitions that are recreated with ${\tt NOSWAP}$:

- Queries that span multiple partitions will not return consistent results across all partitions.
- Queries on the partition that is being swapped will return errors.

- Queries on partitions that are already swapped will be based on the new index.
- Queries on the partitions that haven't been swapped will be based on the old index.

If the *partition_name* is NULL, then this procedure will swap the index metadata. Run this procedure as the last step when recreating a local partitioned index online.

Sync Behavior:

After running <code>EXCHANGE_SHADOW_INDEX</code>, you must call the <code>SYNC_INDEX</code> operation to synchronize any <code>DML</code> that occurs during the build of the shadow index. If you have specified <code>SYNC(ON COMMIT)</code> or <code>SYNC(EVERY)</code>, then the sync occurs automatically. However, if you have specified <code>SYNC(MANUAL)</code>, then you must manually invoke <code>SYNC_INDEX</code>.

Related Topics

```
CTX_DDL."RECREATE_INDEX_ONLINE"
CTX_DDL."CREATE_SHADOW_INDEX"
CTX_DDL."DROP_SHADOW_INDEX"
```

8.36 LOAD_STOPLIST

Use this procedure to load a source file of stopwords to your stoplist for the required language.

A default stoplist is automatically loaded during installation or upgrade based on the database language that you choose. By default, only one SQL file is loaded for the chosen language. You can call the CTX_DDL.LOAD_STOPLIST procedure to customize your stoplist or modify the default list of stopwords. This procedure takes a source file of stopwords for the specified language and adds each word to your stoplist from the stoplist dir/stoplist file.

The Oracle Text supplied stoplists contain default stopwords for all BASIC_LEXER and AUTO LEXER supported languages.

You can also load multi-language stoplists to hold language-specific stopwords. A multi-language stoplist is useful when you index a table that contains documents in different languages, such as English, German, and Japanese. When indexing a multi-language table with a multi-language stoplist, the table must have a language column.

Specify a stoplist in the parameter string of CREATE INDEX or ALTER INDEX to override the default stoplist CTXSYS.DEFAULT_STOPLIST. Add either stopwords, stopclasses, or stopthemes to a stoplist using ADD STOPWORD, ADD STOPCLASS, or ADD STOPTHEME.

Syntax

stoplist name

Specify the name of the stoplist to be loaded.

stoplist dir

Specify the directory location of the source file that you want to load. The source files for default stoplists are located in the *\$ORACLE HOME/ctx/data/stoplist* directory.

stoplist_file

Specify the name of the source file located in the <code>\$ORACLE_HOME/ctx/data/stoplist</code> directory. These source files are named <code>drstopLANG.txt</code>, where <code>LANG</code> specifies the language code. The contents of the source files are the extracted terms from the <code>drdefLANG.sql</code> files (located in the <code>\$ORACLE_HOME/ctx/admin_directory</code>).

language

Specify the language of a stopword when loading multi-language stoplists. You must specify the globalization support name or abbreviation of an Oracle Text-supported language. To make a stopword active in multiple languages, specify ALL for this parameter. For example, defining ALL stopwords is useful when you have international documents that contain English fragments to be stopped in any language. An ALL stopword is active in all languages. If you use multi-lexer, the language-specific lexing of a stopword occurs just as if it has been added multiple times in multiple specific languages. Otherwise, specify NULL.

language_dependent

Set this parameter to FALSE to indicate that any user-defined string can be specified for the language parameter.

Examples

Single-language stoplist:

The following example loads a stoplist named mystop for French (language code: f):

```
begin
ctx_ddl.load_stoplist('mystop', 'ORACLE_HOME/ctx/data/stoplist',
'drstopf.txt');
end;
```

Multi-language stoplist:

The following example loads a multi-language stoplist named multistop for Arabic (language code: ar) and Dutch (language code: nl):

```
begin
ctx_ddl.load_stoplist('multistop', 'ORACLE_HOME/ctx/data/stoplist',
'drstopar.txt', 'arabic');
ctx_ddl.load_stoplist('multistop', 'ORACLE_HOME/ctx/data/stoplist',
'drstopnl.txt', 'dutch');
end;
```

Related Topics

Supplied Stoplists

By default, the system indexes text using the Oracle Text supplied stoplists that correspond to your database language.

CREATE INDEX

Use the CREATE INDEX statement to create an Oracle Text index.

ALTER INDEX

Use the ALTER INDEX statement to change or rebuild an existing index, such as Oracle Text index, Oracle Text search index, JSON search index, XML search index, or hybrid vector index.

8.37 OPTIMIZE_INDEX

Use this procedure to optimize the index. Optimizing an index removes old data and minimizes index fragmentation, which can improve query response time.

Optimize your index after you synchronize it. Querying and DML may proceed while optimization takes place.

You can optimize in fast, full, rebuild, token, token-type, or merge mode.

- Fast mode compacts data but does not remove rows.
- · Full mode compacts data and removes rows.
- Optimize in rebuild mode rebuilds the \$I table (the inverted list table) in its entirety. Rebuilding an index is often significantly faster than performing a full optimization, and is more likely to result in smaller indexes, especially if the index is heavily fragmented.

Rebuild optimization creates a more compact copy of the \$I table, and then switches the original \$I table and the copy. The rebuild operation will therefore require enough space to store the copy as well as the original. (If redo logging is enabled, then additional space is required in the redo log as well.) At the end of the rebuild operation, the original \$I table is dropped, and the space can be reused. A temporary "change capture trigger" is used to ensure that updates to the \$I table during the optimization are not lost. For this reason, the user calling OPTIMIZE INDEX in REBUILD mode must have the CREATE TRIGGER privilege.

Optimize in rebuild mode supports partitioning on the \$I table via the i_table_clause attribute of the basic storage preference with the following limitations:

- The i_index_clause must specify using a local btree index if the \$I table is partitioned.
- Partitioning schemes on the token_first, token_last, or token_count columns are not allowed.
- In token mode, specify a specific token to be optimized (for example, all rows with documents containing the word *elections*). Use this mode to optimize index tokens that are frequently searched, without spending time on optimizing tokens that are rarely referenced. An optimized token can improve guery response time (but only for gueries on that token).
 - Starting with Oracle Database Release 21c, the topN fragmented tokens in the \$I table are optimized. The \$SN and \$ST tables are also optimized.
- Token-type optimization is similar to token mode, except that the optimization is performed
 on field, MDATA, or SDATA sections (for example, sections with an <A> tag). This is useful in
 keeping critical field or MDATA sections optimal.
- Use the merge mode to optimize the \$I table for the CONTEXT indexes that are frequently
 used for DML operations. The merge operation compacts the existing data in the \$G table,
 and then copies that data to the \$I table. The compacted rows are then deleted from
 the \$G table.

A common strategy for optimizing indexes is to perform regular token optimizations on frequently referenced terms, and to perform rebuild optimizations less frequently. (Use CTX_REPORT.QUERY_LOG_SUMMARY to find out which queries are made most frequently.) You can perform full, fast, or token-type optimizations instead of token optimizations.



Some users choose to perform frequent time-limited full optimizations along with occasional rebuild optimizations.



- Optimizing an index can result in better response time only if you insert, delete, or update documents in the base table after your initial indexing operation.
- When you run index optimization, any work in the session is committed effectively and that work cannot be rolled back.
- You cannot run DDL ALTER TABLE .. MODIFY concurrently with an index synchronization or index maintenance operation, such as SYNC INDEX.

Using this procedure to optimize the index is recommended over using the ALTER INDEX statement.

Optimization of a large index may take a long time. To monitor the progress of a lengthy optimization, log the optimization with CTX_OUTPUT.START_LOG and check the resultant logfile from time to time.

Note that, unlike serial optimize full, CTX_DDL.OPTIMIZE_INDEX() run with optlevel of FULL and parallel_degree > 1 is not resumable. That is, it will not resume from where it left after a time-out or failure.

Note:

There is a very small window of time when a query might fail in CTX_DDL.OPTIMIZE_INDEX REBUILD mode when the \$I table is being swapped with the optimized shadow \$I table.

Syntax

idx name

Specify the name of the index. If you do not specify an index name, then Oracle Text chooses a single index to optimize.



optlevel

Specify optimization level as a string. You can specify one of the following methods for optimization:

optlevel value	Description
FAST or CTX_DDL.OPTLEVEL_FAST	This method compacts fragmented rows. However, old data is not removed. FAST optimization is not supported for CTXCAT indexes. FAST optimization will not optimize \$S index table. Fast optimization is not supported for local Oracle Text search indexes.
FULL or CTX_DDL.OPTLEVEL_FULL	In this mode you can optimize the entire index or a portion of the index. This method compacts rows and removes old data (deleted rows). Optimizing in full mode runs even when there are no deleted rows. Full optimization is not supported for CTXCAT indexes.
REBUILD or CTX_DDL.OPTLEVEL_REBUILD	This optlevel rebuilds the \$I table (the inverted list table) to produce more compact token info rows. Like FULL optimize, this mode also deletes information pertaining to deleted rows of the base table. REBUILD is not supported for CTCAT, CTXRULE, and local Oracle Text search indexes.
TOKEN or CTX_DDL.OPTLEVEL_TOKEN	This method lets you specify a specific token to be optimized. Oracle Text does a full optimization on the token you specify with token. If no token type is provided, 0 (zero) will be used as the default. Use this method to optimize those tokens that are searched frequently. Token optimization is not supported for CTCAT and CTXRULE indexes.
TOKEN_TYPE or CTX_DDL.OPTLEVEL_TOKEN_TYPE	This optlevel optimizes on demand all tokens in the index matching either the input token type or the input section type. When optlevel is TOKEN_TYPE, either token_type or section_type must be provided. TOKEN_TYPE performs FULL optimize on any token of the input token_type or section_type, whichever is provided. Like a TOKEN optimize, TOKEN_TYPE optimize does not change the FULL optimize state, and runs to completion on each invocation. Token_type optimization is not supported for CTCAT and CTXRULE indexes.
MERGE or CTX_DDL.OPTLEVEL_MERGE	This optlevel compacts the rows in the staging \$G table and merges them into the \$I table. This option is not supported at the token level. Specifying the TOKEN attribute with this option results in an error. Merge optimization should be used for CONTEXT indexes with the STAGE_ITAB index option enabled.
The behavior of CTX_DDL.OPT	IMIZE_INDEX with respect to the \$S index table is as follows:
C	Will Notes Optimize \$S ndex Table Yes/No
FAST or CTX_DDL.OPTLEVEL_FAST	No



optlevel value	Will Optimize \$S Index Table Yes/No	Notes
FULL or CTX_DDL.OPTLEVEL_FULL	Yes	 The optimize process will optimize \$I table first. Once \$I table optimize is finished, CTX_DDL.OPTIMIZE_INDEX will continue on to optimize \$S index table. MAXTIME will also be honored. Once CTX_DDL.OPTIMIZE_INDEX completes optimizing \$S rows for a given SDATA_ID, it will check MAXTIME and exit if total elapsed time (including time taken to optimize \$I) exceeds specified MAXTIME. The next CTX_DDL.OPTIMIZE_INDEX with optlevel=>'FULL' will pick up where it left off.
		 \$S table optimize will be done in serial.
REBUILD or CTX_DDL.OPTLEVEL_REBUILD	Yes	 \$S optimize will start after \$I rebuild finishes. \$S optimize in this case will be processed the same way as \$S optimize in FULL mode. \$S table is optimized in place, not rebuilt. Note: If for some reason \$S optimize exits unusually, then it is recommended that you use optlevel=>TOKEN_TYPE to optimize \$S to avoid rebuilding the \$I table again. \$S table optimize will be done in serial.
TOKEN or CTX_DDL.OPTLEVEL_TOKEN	No	
TOKEN_TYPE or CTX_DDL.OPTLEVEL_TOKEN_TYPE	Yes	You can optimize \$S rows for a given SDATA_ID by setting optlevel => TOKEN_TYPE and the TOKEN_TYPE parameter to the target SDATA_ID.

maxtime

Specify maximum optimization time, in minutes, for ${\tt FULL}$ optimize.

When you specify the symbol CTX_DDL.MAXTIME_UNLIMITED (or pass in NULL), the entire index is optimized. This is the default.

token

Specify the token to be optimized.

part_name

If your index is a local index, then you must specify the name of the index partition to synchronize otherwise an error is returned.

If your index is a global, nonpartitioned index, then specify NULL, which is the default.

token type

Specify the token type to be optimized.

You can find the token_type by using the CTX_REPORT.TOKEN_TYPE method or the CTX USER SECTIONS view.

parallel_degree

Specify the parallel degree as a number for parallel optimization. The actual parallel degree depends on your resources.



Because the optlevel values are executed serially, this setting throws the error DRG-10598 for the following values:

- TOKEN Or CTX_DDL.OPTLEVEL_TOKEN
- FAST Or CTX DDL.OPTLEVEL FAST

maxtokens

Specify the maxtokens to be optimized.

maxtokens attribute can be specified only when optlevel value is set to TOKEN or CTX DDL.OPTLEVEL TOKEN and when the token parameter is NULL.

Note:

- If the number of fragmented tokens exceeds 50% of total number of tokens in \$I and maxtokens is not specified, then "index too fragmented" error is returned.
- If maxtokens specified is negative or greater than 50% of total number of tokens in \$1, then "invalid value for maxtokens" error is returned.

section_type

Specify the <code>section_type</code> to optimize all sections of a certain type. This parameter can have one of the following values:

section_type value	Description
CTX_DDL.SECTION_FIELD	The optimization is run for all field sections in the index.
CTX_DDL.SECTION_SORT_SDATA	The optimization is run for all <code>optimized_for</code> sort SDATA sections in the index.
CTX_DDL.SECTION_MDATA	The optimization is run for all MDATA sections in the index.
CTX_DDL.SECTION_SEARCH_SDATA	The optimization is run for all <code>optimized_for</code> search SDATA sections in the index.
CTX_DDL.SECTION_WILDCARD_INDEX	The optimization is run for the \$KG table (that is, the wildcard search index).

Note:

- You can specify section_type only when the optlevel value is set to TOKEN TYPE or CTX DDL.OPTLEVEL TOKEN TYPE.
- In the absence of sections of the specified type, index optimization for section_type is a no-op (no operations). Similarly, in the absence of the \$KG table, index optimization for the CTX_DDL.SECTION_WILDCARD_INDEX section type value is a no-op.

Examples

The following two examples are equivalent ways of optimizing an index using fast optimization:

```
begin
  ctx_ddl.optimize_index('myidx','FAST');
end:
```



```
begin
  ctx_ddl.optimize_index('myidx',CTX_DDL.OPTLEVEL_FAST);
end;
```

The following example optimizes the index token *Oracle*:

```
begin
   ctx_ddl.optimize_index('myidx','token', TOKEN=>'Oracle');
end;
```

To optimize all tokens of field section MYSEC in index MYINDEX:

```
begin
  ctx_ddl.optimize_index('myindex', ctx_ddl.optlevel_token_type,
     token type=> ctx report.token type('myindex','field mysec text'));end;
```

The following two examples are equivalent ways of optimizing an index using merge optimization:

```
begin
   ctx_ddl.optimize_index('idx','MERGE');
end;

begin
   ctx_ddl.optimize_index('idx',CTX_DDL.OPTLEVEL_MERGE);
end:
```

The following example optimizes the top 10 fragmented tokens in \$1:

```
begin
  ctx_ddl.optimize_index('idx','TOKEN',maxtokens=>10);
end;
```

Notes

- You can run CTX_DDL.SYNC_INDEX and CTX_DDL.OPTIMIZE_INDEX at the same time. You can also run CTX_DDL.SYNC_INDEX and CTX_DDL.OPTIMIZE_INDEX with parallelism at the same time. However, you should not:
 - Run CTX_DDL.SYNC_INDEX with parallelism at the same time as CTX_DDL.OPTIMIZE_INDEX
 - Run CTX_DDL.SYNC_INDEX with parallelism at the same time as
 CTX DDL.OPTIMIZE INDEX with parallelism.

If you should run one of these combinations, no error is generated; however, one operation will wait until the other is done.

• You cannot sync or optimize an index that is owned by a different schema. Doing so results in the "DRG-10016: You must be the owner to modify this object" error.

Related Topics

SYNC INDEX

Synchronizes the index to process inserts, updates, and deletes to the base table.

ALTER INDEX

Use the ALTER INDEX statement to change or rebuild an existing index, such as Oracle Text index, Oracle Text search index, JSON search index, XML search index, or hybrid vector index.



8.38 POPULATE_PENDING

This procedure populates the pending queue with every ROWID in the base table or table partition. This procedure is only supported for CONTEXT indexes.

This procedure is valuable for large installations that cannot afford to have the indexing process run continuously, and, therefore, need finer control over creating text indexes. The preferred method is to create an empty index, place all the ROWIDs into the pending queue, and build the index through CTX DDL.SYNC INDEX.

Syntax

```
ctx_ddl.populate_pending(
  idx_name     IN VARCHAR2,
  part_name     IN VARCHAR2 DEFAULT NULL
);
```

idx name

Name of the CONTEXT indextype.

part_name

Name of the index partition, if any. Must be provided for local partitioned indexes and must be NULL for global, nonpartitioned indexes.

Notes

The SYNC_INDEX is blocked for the duration of the processing. The index unit must be totally empty (idx docid count = 0, idx nextid = 1).

The ROWIDs of rows that are waiting for synchronization into the index are inserted into the DRINDEX_NAMEC table. You should ensure that there is sufficient space in these tables to hold the ROWIDs of the base table.

Related Topics

- Oracle Text Application Developer's Guide
- SYNC_INDEX
 Synchronizes the index to process inserts, updates, and deletes to the base table.

8.39 PREFERENCE IMPLICIT COMMIT

This variable, set at the package level for CTX_DDL, determines whether procedures related to CTX_DDL preferences issue an implicit commit and is session duration.

You can set the PREFERENCE_IMPLICIT_COMMIT variable for the procedures listed in the following table.

Procedure Name	Procedure Name
ADD_ATTR_SECTION	CREATE_INDEX_SET
ADD_FIELD_SECTION	CREATE_PREFERENCE
ADD_INDEX	CREATE_SECTION_GROUP
ADD_MDATA_COLUMN	CREATE_STOPLIST
ADD_MDATA_SECTION	DROP_PREFERENCE

Procedure Name	Procedure Name
ADD_SDATA_COLUMN	DROP_SECTION_GROUP
ADD_SDATA_SECTION	DROP_STOPLIST
ADD_SPECIAL_SECTION	REMOVE_INDEX
ADD_STOPCLASS	REMOVE_SECTION
ADD_STOP_SECTION	REMOVE_SUB_LEXER
ADD_STOPTHEME	SET_ATTRIBUTE
ADD_STOPWORD	UNSET_ATTRIBUTE
ADD_SUB_LEXER	UPDATE_SUB_LEXER
ADD_ZONE_SECTION	

Note:

The REMOVE_STOPCLASS, REMOVE_STOPTHEME, and REMOVE_STOPWORD procedures do not issue an implicit commit, and, therefore, do not use the PREFERENCE IMPLICIT COMMIT flag.

Syntax

exec CTX DDL.PREFERENCE IMPLICIT COMMIT := TRUE | FALSE ;

The default value of the PREFERENCE_IMPLICIT_COMMIT variable is TRUE. When this variable is set to FALSE, procedures related to CTX_DDL preferences will not issue an implicit commit. This enables you to easily rollback multiple preference changes. This variable is session duration.

Example

The following example turns off implicit commit.

exec CTX DDL.PREFERENCE IMPLICIT COMMIT : update sub lexer = FALSE;

8.40 RECREATE_INDEX_ONLINE

Recreates the specified index, or recreates the passed-in index partition if the index is local partitioned.

For global nonpartitioned indexes, this is a one-step procedure. For local partitioned indexes, this procedure must be run separately on every partition after first using CREATE_SHADOW_INDEX to create a shadow policy (or metadata). This procedure is only supported in Enterprise Edition of Oracle Database.

The following changes are not supported:

- Transitioning from non-composite domain index to composite, or changing the composite domain index columns.
- Rebuilding indexes that have partitioned index tables, for example, \$I, \$P, \$K.



Syntax

idx name

The name of a valid CONTEXT indextype.

parameter string

If the index is a global nonpartitioned index, specify the same index-level parameter string as in ALTER INDEX. Must start with REPLACE, if it is not NULL. Optionally specify SWAP or NOSWAP. The default is SWAP.

parallel_degree

Reserved for future use. Specify the degree of parallelism. Parallel operation is not supported in the current release.

partition name

Specify the name of a valid index partition for a local partitioned index. Otherwise, the default is <code>NULL</code>. If the index is partitioned, then first pass a partition name, and then specify the partition-level parameter string for <code>ALTER INDEX REBUILD PARTITION</code>.

Examples

Example 8-5 Recreate Simple Global Index

The following example creates an index <code>idx</code> with a <code>BASIC_LEXER-based</code> preference <code>us_lexer</code>. It then recreates the index with a new <code>MULTI_LEXER</code> based preference <code>m_lexer</code> in one step. You can use this one step approach when you do not mind that a query might fail for a small window of time at the end of the operation, and DML might get blocked at the beginning for a short time and again at the end.

```
/* create lexer and original index */
exec ctx ddl.create preference('us lexer', 'basic lexer');
create index idx on tbl(text) indextype is ctxsys.context
 parameters('lexer us lexer');
/* create a new lexer */
 ctx ddl.create preference('e lexer', 'basic lexer');
 ctx_ddl.set_attribute('e_lexer', 'base_letter', 'yes');
 ctx_ddl.create_preference('m_lexer','multi_lexer');
 ctx_ddl.add_sub_lexer('m_lexer','default','us_lexer');
 ctx ddl.add sub lexer('m lexer','e','e lexer');
end;
/* add new language column to the table for multi-lexer */
alter table tbl add(lang varchar2(10) default 'us');
/* recreate index online with the new multip-lexer */
exec ctx ddl.recreate index online('idx',
  'replace lexer m lexer language column lang');
```



Example 8-6 Local Index Recreate with All-At-Once Swap

The following example creates a local partitioned index idxp with a basic lexer us_lexer . It has two index partitions idx_p1 and idx_p2 . It then recreates a local partitioned index idxp online with partition idx_p1 , which will have a new storage preference new_store . The swapping of the partition metadata and index partition data occur at the end. In this example, queries spanning multiple partitions return consistent results across partitions when recreate is in process, except at the end when EXCHANGE_SHADOW_INDEX is running. The extra space required is the combined index size of partition idx_p1 and idx_p2 .

```
/* create a basic lexer and a local partition index with the lexer*/
exec ctx ddl.create preference('us lexer', 'basic lexer');
create index idxp on tblp(text) indextype is ctxsys.context local
  (partition idx p1,
  partition idx p2)
 parameters('lexer us lexer');
/* create new preferences */
begin
  ctx ddl.create preference('my_store','basic_storage');
  ctx ddl.set attribute('my store','i table clause','tablespace tbs');
end;
begin
  ctx ddl.create preference('e lexer', 'basic lexer');
  ctx ddl.set attribute('e lexer', 'base letter', 'yes');
  ctx ddl.create preference('m lexer', 'multi lexer');
  ctx ddl.add sub lexer('m lexer','default','us lexer');
  ctx ddl.add sub lexer('m lexer','e','e lexer');
end;
/* add new language column */
alter table tblp add column (lang varchar2(10) default 'us');
/* create a shadow policy with a new lexer */
exec ctx ddl.create shadow index('idxp', null,
  'replace lexer m lexer language column lang');
/* recreate every index partition online without swapping */
exec ctx ddl.recreate index online('idxp',
   'replace storage my_store NOSWAP', 1, 'idx p1');
exec ctx ddl.recreate index online('idxp','replace NOSWAP',1,'idx p2');
/* exchange in shadow index partition all at once */
exec ctx ddl.exchange shadow index('idxp',
      'idx p1') /* exchange index partition data*/
exec ctx ddl.exchange shadow index('idxp',
      'idx p2') /* exchange index partition data*/
/* exchange in shadow index metadata */
exec ctx ddl.exchange shadow index('idxp')
```

Example 8-7 Local Index Recreate with Per-Partition Swap

This example performs the same tasks as Example 8-6, except that each index partition is swapped in as it is completed. Queries across all partitions may return inconsistent results in this example.

```
/* create a basic lexer and a local partition index with the lexer*/
exec ctx_ddl.create_preference('us_lexer','basic_lexer');
create index idxp on tblp(text) indextype is ctxsys.context local
```

```
(partition idx p1,
  partition idx p2)
 parameters('lexer us_lexer');
/* create new preferences */
begin
  ctx ddl.create preference('my store', 'basic storage');
  ctx_ddl.set_attribute('my_store','i_table_clause','tablespace tbs');
end;
begin
  ctx_ddl.create_preference('e_lexer','basic_lexer');
  ctx_ddl.set_attribute('e_lexer', 'base_letter', 'yes');
  ctx ddl.create preference('m lexer', 'multi lexer');
  ctx ddl.add sub lexer('m lexer', 'default', 'us lexer');
  ctx ddl.add sub lexer('m lexer','e','e lexer');
end:
/* add new language column */
alter table tblp add column (lang varchar2(10) default 'us');
/* create a shadow policy with a new lexer *
exec ctx_ddl.create_shadow index('idxp',
  'replace lexer m lexer language column lang');
/* recreate every index partition online and swap (default) */
exec ctx ddl.recreate index online('idxp',
       'replace storage my store', 1, 'idx p1');
exec ctx_ddl.recreate_index_online('idxp', 'replace SWAP', 1, 'idx p2',
    /* exchange in shadow index metadata */
exec ctx_ddl.exchange_shadow_index('idxp')
```

Example 8-8 Scheduled Local Index Recreate with All-At-Once Swap

This example shows the incremental recreation of a local partitioned index, where partitions are all swapped at the end.

```
/* create a basic lexer and a local partition index with the lexer*/
exec ctx ddl.create preference('us_lexer','basic_lexer');
create index idxp on tblp(text) indextype is ctxsys.context local
  (partition idx p1,
   partition idx p2)
parameters('lexer us_lexer');
/* create new preferences */
  ctx ddl.create preference('my store', 'basic storage');
  ctx_ddl.set_attribute('my_store','i_table_clause','tablespace tbs');
end;
begin
  ctx_ddl.create_preference('e_lexer','basic_lexer');
  ctx_ddl.set_attribute('e_lexer', 'base_letter', 'yes');
  ctx_ddl.create_preference('m_lexer','multi_lexer');
  ctx_ddl.add_sub_lexer('m_lexer','default','us_lexer');
  ctx ddl.add sub lexer('m lexer','e','e lexer');
end;
/* add new language column */
alter table tblp add column (lang varchar2(10) default 'us');
```

```
/* create a shadow policy with a new lexer *
exec ctx_ddl.create_shadow_index('idxp',
 'replace lexer m lexer language column lang');
/* create shadow partition with new storage preference */
exec ctx ddl.recreate index online('idxp', 'replace storage ctxsys.default storage
nopopulate',1,'idx p1');
exec ctx ddl.recreate index online('idxp', 'replace storage ctxsys.default storage
nopopulate',1,'idx p2');
declare
 idxid integer;
 ixpid integer;
 select idx id into idxid from ctx user indexes
   where idx name = 'IDXP';
 select ixp id into ixpid from ctx_user_index_partitions
   where ixp index name = 'IDXP'
         and ixp index partition name = 'IDX P1';
  /* populate pending */
 ctx ddl.populate pending('RIO$'||idxid, 'RIO$'||idxid||'#'||ixpid);
  /* incremental sync
 ctx ddl.sync index('RIO$'||idxid, null, 'RIO$'||idxid||'#'||ixpid,
                     maxtime=>400);
 /* more incremental sync until no more pending rows */
 select ixp id into ixpid from ctx user index partitions
   where ixp_index_name = 'IDXP'
          and ixp index partition name = 'IDX P2';
  /* populate pending */
 ctx ddl.populate pending('RIO$'||idxid, 'RIO$'||idxid||'#'||ixpid);
  /* incremental sync
 ctx_ddl.sync_index('RIO$'||idxid, null, 'RIO$'||idxid||'#'||ixpid,
                     maxtime=>400);
 /* more incremental sync until no more pending rows */
end;
/
exec ctx ddl.exchange shadow index('idxp','idx p1');
exec ctx ddl.exchange shadow index('idxp','idx p2');
exec ctx ddl.exchange shadow index('idxp');
```

Example 8-9 Schedule Local Index Recreate with Per-Partition Swap

For incremental recreate where partitions are swapped as they becomes available, follow the steps in example Example 8-8, except instead of waiting until all syncs are finished before starting exchange shadow index, EXCHANGE_SHADOW_INDEX is done for each partition right after sync is finished.

Notes

Using RECREATE_INDEX_ONLINE with Global Nonpartitioned Indexes:

For global indexes, this procedure provides a one-step process to recreate an index online. It recreates an index, with new preference values, while preserving base table DML and query capability during the recreate process.

Because the new index is created alongside the existing index, this operation requires additional storage roughly equal to the size of the existing index.

DML Behavior:

Because this procedure is performed online, DML on the base table are permitted during this operation, and are processed as normal. All DML statements that occur during RECREATE INDEX ONLINE are logged into an online pending queue.

Towards the end of the recreate operation, there will be a short duration when DML will fail with an error being raised stating that the index is in an in-progress status. DML may stop responding again during the process, and the duration will depend on how many DML are logged in the online pending queue since the start of the recreate process.

Note that after the recreate index operation is complete, new information, from all the DML that becomes pending since <code>RECREATE_INDEX_ONLINE</code> started, may not be immediately reflected. As with creating an index with <code>INDEXTYPE IS ctxsys.context ONLINE</code>, the index should be synchronized after the recreate index operation is complete, to bring it fully upto-date.

Sync and Optimize Behavior:

Syncs issued against the index during the recreate operation are processed against the old, existing data. Syncs are also blocked during the same window when queries return errors.

After running RECREATE_INDEX_ONLINE, you must call the SYNC_INDEX operation to synchronize any DML that occurs during the build of the shadow index. If you have specified SYNC (ON COMMIT) or SYNC (EVERY), then the sync occurs automatically. However, if you have specified SYNC (MANUAL), then you must manually invoke SYNC INDEX.

Optimize commands issued against the index during the recreate operation return immediately without error and without processing.

Query Behavior:

During the recreate operation, the index can be queried normally most of the time. Queries return results based on the existing index and policy (or metadata) until after the final swap.

There is a short interval towards the end of RECREATE_INDEX_ONLINE when queries will return an error indicating that the column is not indexed. This duration should be short for regular queries. It is mainly the time taken for swapping data segments of the shadow index tables and the index tables, plus the time to delete all the rows in the pending queue. This is the same window of time when DML will fail.

During RECREATE_INDEX_ONLINE, if you issue DML statements and synchronize them, then you will be able to see the new rows when you query on the existing index. However, after RECREATE_INDEX_ONLINE finishes (swapping completes and query is on the new index) and before sync is performed, it is possible that you will not be able to query on the new rows, which once could be queried on the old index.

Transactional queries are not supported.

Using RECREATE_INDEX_ONLINE with Local Partitioned Indexes:

If the index is local partitioned, you cannot recreate index in one step. You must first create a shadow policy, and then run this procedure for every partition. You can specify SWAP or NOSWAP to indicate whether RECREATE_INDEX_ONLINE partition will swap the index partition data and index partition metadata or not. If the partition was built with NOSWAP, then another call to EXCHANGE SHADOW INDEX must be invoked later against this partition.

This procedure can also be used to update the metadata (for example, storage preference) of each partition when you specify NOPOPULATE in the parameter string. This is useful for incremental building of a shadow index through time-limited sync.

If NOPOPULATE is specified, then NOSWAP is silently enforced.



NOSWAP Behavior:

During the recreate of the index partition, since no swapping is performed, queries on the partition are processed regularly. Until the swapping stage is reached, queries spanning multiple partitions return consistent results across partitions.

DML and sync are processed normally. Running optimize on partitions that are being recreated, or that have been built (but not swapped), simply returns without doing anything. Running optimize on a partition that has not been rebuilt processes normally.

As with a global index, when all of the partitions use NOSWAP, the additional storage requirement is roughly equal to the size of the existing index.

SWAP Behavior:

Because index partition data and metadata are swapped after index recreate, queries that span multiple partitions will not return consistent results from partition to partition, but will always be correct with respect to each index partition. There is also a short interval towards the end of partition recreate, when the index partition is swapped, during which a query will return a "column not indexed" error.

When partitions are recreated with SWAP, the additional storage requirement for the operation is equal to the size of the existing index partition.

DML on the partition is blocked. Sync is also blocked during swapping.

Restrictions:

The RECREATE_INDEX_ONLINE and CREATE_SHADOW_INDEX procedures do not support section group with the XML_ENABLE attribute on CONTEXT indexes. Doing so results in the "DRG-10521: Operation not supported with XML ENABLE on a CONTEXT Index" error.

The RECREATE_INDEX_ONLINE and CREATE_SHADOW_INDEX procedures are not supported for search indexes.

Related Topics

CREATE INDEX

Use the CREATE INDEX statement to create an Oracle Text index.

CREATE SHADOW INDEX

Creates index metadata (or policy) for the specified index. If the index is not partitioned, then it also creates the index tables. This procedure is only supported in Enterprise Edition of Oracle Database.

EXCHANGE SHADOW INDEX

This procedure swaps the index (or index partition) metadata and index (or index partition) data.

- DROP SHADOW INDEX
- Oracle Text Application Developer's Guide

8.41 REM SEC GRP ATTR VAL

Removes a specific section group attribute value from the list of values of an existing section group attribute.

Syntax

```
CTX_DDL.REM_SEC_GRP_ATTR_VAL(group_name IN VARCHAR2, attribute_name IN VARCHAR2, attribute value IN VARCHAR2);
```



group_name

Specify the section group name.

attribute name

Specify the name of the section group attribute.

attribute_value

Specify the section group attribute value.

8.42 REMOVE_AUTO_OPTIMIZE

Removes an index or partition from the list of indexes subject to auto optimization. No new auto optimization calls are made to this index. The removal takes effect immediately.

If the specified index is not in the existing list of indexes, then an error occurs. For partitioned indexes, an error occurs when the partition name is not specified.

Note:

In Oracle Database Release 21c, the procedures <code>ADD_AUTO_OPTIMIZE</code> and <code>REMOVE_AUTO_OPTIMIZE</code>, and the views <code>CTX_AUTO_OPTIMIZE_INDEXES</code>, <code>CTX_USER_AUTO_OPTIMIZE_INDEXES</code> and <code>CTX_AUTO_OPTIMIZE_STATUS</code> are deprecated.

Syntax

idx_name

Specify the name of the index to remove.

part_name

Specify the name of the partition to remove.

Related Topic

"ADD_AUTO_OPTIMIZE"

8.43 REMOVE_INDEX

Removes the index with the specified column list from a CTXCAT index set preference.

This procedure does not remove a CTXCAT sub-index from the existing index. To do so, you must drop your index and re-index with the modified index set preference.



The Oracle Text indextype CTXCAT is deprecated with Oracle Database 23ai. The indextype itself, and it's operator CTXCAT, can be removed in a future release. CTXCAT was introduced when indexes were typically a few megabytes in size. Modern, large indexes, can be difficult to manage with CTXCAT. The addition of index sets to CTXCAT can be achieved more effectively by the use of FILTER BY and ORDER BY columns, or SDATA, or both, in the CONTEXT indextype. CTXCAT is therefore rarely an appropriate choice. Oracle recommends that you choose the more efficient CONTEXT indextype.

Syntax

set name

Specify the name of the index set.

column list

Specify the name of the column list to remove.

8.44 REMOVE MDATA

Use this procedure to remove metadata values, which are associated with an MDATA section, from a document.

Only the owner of the index is allowed to call ADD_MDATA and REMOVE MDATA.

CTX_DDL.REMOVE_MDATA is transactional and takes effect immediately in the calling session. This procedure can be seen only in the calling session and must be committed to take permanent effect. You can reverse this procedure with a ROLLBACK command.

Syntax

idx name

Name of the text index that contains the named *rowids*.

section name

Name of the MDATA section.

values

List of metadata values. If a metadata value contains a comma, the comma must be escaped with a backslash.

rowids

Rowids from which to remove the metadata values.

[part_name]

Name of the index partition, if any. Must be provided for local partitioned indexes and must be NULL for global, nonpartitioned indexes.

Example

This example removes the MDATA value blue from the MDATA section BGCOLOR.

```
ctx_ddl.remove_mdata('idx_docs', 'bgcolor', 'blue', 'rows');
```

Note:

- These updates are updates directly on the index itself, not on the actual contents stored in the base table. Therefore, they will not exist when the Text index is rebuilt.
- CTX_DDL.REMOVE_MDATA is not supported for documents with Oracle Text search index as stage itab is ON by default for Oracle Text search index.

Related Topics

```
"ADD_MDATA"

"ADD_MDATA_SECTION"
```

"MDATA"

The Section Searching chapter of Oracle Text Application Developer's Guide

8.45 REMOVE_SECTION

The REMOVE_SECTION procedure removes the specified section from the specified section group. You can specify the section by name or ID. View section ID with the CTX_USER_SECTIONS view.

Syntax 1

Use the following syntax to remove a section by section name:

group_name

Specify the name of the section group from which to delete section_name.

section_name

Specify the name of the section to delete from group name.

Syntax 2

Use the following syntax to remove a section by section ID:

```
CTX_DDL.REMOVE_SECTION(
  group_name     IN      VARCHAR2,
  section_id     IN      NUMBER
);
```

group_name

Specify the name of the section group from which to delete section id.

section id

Specify the section ID of the section to delete from group name.

Example

The following example drops a section called Title from the htmgroup:

```
begin
ctx_ddl.remove_section('htmgroup', 'Title');
end;
```

Related Topics

```
"ADD_FIELD_SECTION"

"ADD_SPECIAL_SECTION"

"ADD ZONE SECTION"
```

8.46 REMOVE_STOPCLASS

Removes a stopclass from a stoplist.

Syntax

stoplist_name

Specify the name of the stoplist.

stopclass

Specify the name of the stopclass to be removed.

Example

The following example removes the stopclass NUMBERS from the stoplist mystop.

```
begin
ctx_ddl.remove_stopclass('mystop', 'NUMBERS');
end;
```



Related Topic

"ADD_STOPCLASS"

8.47 REMOVE_STOPTHEME

Removes a stoptheme from a stoplist.

Syntax

stoplist_name

Specify the name of the stoplist.

stoptheme

Specify the stoptheme to be removed from stoplist name.

Example

The following example removes the stoptheme *banking* from the stoplist mystop:

```
begin
ctx_ddl.remove_stoptheme('mystop', 'banking');
end;
```

Related Topic

"ADD_STOPTHEME"

8.48 REMOVE_STOPWORD

Removes a stopword from a stoplist. To have the removal of a stopword be reflected in the index, you must rebuild your index. You can also remove a language-independent stopword.

Syntax

```
CTX_DDL.REMOVE_STOPWORD(

stoplist_name IN VARCHAR2,
stopword IN VARCHAR2,
language IN VARCHAR2 default NULL
);
```

stoplist_name

Specify the name of the stoplist.

stopword

Specify the stopword to be removed from stoplist name.



language

Specify the language of stopword to remove when the stoplist you specify with stoplist_name is of type MULTI_STOPLIST. You must specify the globalization support name or abbreviation of an Oracle Text-supported language. You can also remove ALL stopwords.

Example

The following example removes a stopword because from the stoplist mystop:

```
begin
ctx_ddl.remove_stopword('mystop','because');
end;
```

Related Topic

"ADD_STOPWORD"

8.49 REMOVE_SUB_LEXER

Removes a sub-lexer from a multi-lexer preference. You cannot remove the lexer for DEFAULT. You can also remove a language-independent sub-lexer.

Syntax

```
CTX_DDL.REMOVE_SUB_LEXER(
lexer_name IN VARCHAR2,
language IN VARCHAR2 default NULL
):
```

lexer_name

Specify the name of the multi-lexer preference or language-independent sub-lexer.

language

Specify the language of the sub-lexer to remove. You must specify the globalization support name or abbreviation of an Oracle Text-supported language.

Example

The following example removes a sub-lexer *german_lexer* of language *german*:

```
begin
ctx_ddl.remove_sub_lexer('german_lexer','german');
end;
```

Related Topic

"ADD_SUB_LEXER "

8.50 REPLACE_INDEX_METADATA

Use this procedure to replace metadata in local domain indexes at the global (index) level.

Note:

The ALTER INDEX PARAMETERS command performs the same function as this procedure and can replace more than just metadata. For that reason, using ALTER INDEX PARAMETERS is the preferred method of replacing metadata at the global (index) level and should be used in place of this procedure when possible. For more information, see "ALTER INDEX PARAMETERS Syntax".

CTX REPLACE INDEX METADATA may be deprecated in a future release of Oracle Text.

Syntax

idx name

Specify the name of the index whose metadata you want to replace.

parameter_string

Specify the parameter string to be passed to ALTER INDEX. This must begin with 'REPLACE METADATA'.

Notes

ALTER INDEX REBUILD PARAMETERS ('REPLACE METADATA') does not work for a local partitioned index at the index (global) level. You cannot, for example, use that ALTER INDEX syntax to change a global preference, such as filter or lexer type, without rebuilding the index. Therefore, CTX_DDL.REPLACE_INDEX_METADATA is provided as a method of overcoming this limitation of ALTER INDEX. Also, ALTER INDEX REBUILD PARAMETERS ('REPLACE METADATA') does not work with forward index; instead use 'REPLACE STORAGE'.

Though it is meant as a way to replace metadata for a local partitioned index, CTX DDL.REPLACE INDEX METADATA can be used on a global, nonpartitioned index, as well.

REPLACE_INDEX_METADATA cannot be used to change the sync type at the partition level; that is, parameter_string cannot be 'REPLACE METADATA SYNC'. For that purpose, use ALTER INDEX REBUILD PARTITION to change the sync type at the partition level.

Related Topics

"ALTER INDEX PARAMETERS Syntax"

"ALTER INDEX REBUILD Syntax"

8.51 SET_ATTRIBUTE

Sets a preference attribute. Use this procedure after you have created a preference with ${\tt CTX}\ {\tt DDL.CREATE}\ {\tt PREFERENCE}.$

Syntax



```
attribute_value IN VARCHAR2
```

);

preference_name

Specify the name of the preference.



Procedure names should not include the semicolon character.

attribute name

Specify the name of the attribute.

attribute value

Specify the attribute value. Specify boolean values as TRUE or FALSE, T or F, YES or NO, Y or N, ON or OFF, or 1 or 0.

Examples

Example 8-10 Specifying File Data Storage

The following example creates a data storage preference called filepref that tells the system that the files to be indexed are stored in an Oracle directory object. The example then uses CTX_DDL.SET_ATTRIBUTE to set the DIRECTORY attribute to the directory /docs.

```
begin
ctx_ddl.create_preference('filepref', 'DIRECTORY_DATASTORE');
ctx_ddl.set_attribute('filepref', 'DIRECTORY', '/docs');
end;
```

Example 8-11 Storing Text Index Tables in the In-Memory Column Store

This example creates a storage preference called <code>mysto</code> of type <code>BASIC_STORAGE</code> that specifies that the <code>\$I</code> index table must be stored in the In-Memory Column Store (IM column store).

```
exec ctx_ddl.create_preference('mysto', 'basic_storage');
exec ctx_ddl.set_attribute('mysto', 'I_TABLE_CLAUSE', 'inmemory');
```

Related Topics

- CREATE_PREFERENCE
 Creates a preference in the Text data dictionary.
- Datastore Types

Use the datastore types to create a datastore preference. This helps you specify how your text is stored.

BASIC_STORAGE

8.52 SET SEC GRP ATTR

Adds a section group-specific attribute to a section group identified by name.

Also used to set xml enable to support XML awareness.

Syntax

group_name

Specify the name of the section group.

attribute_name

Specify the name of the section group attribute.

attribute_value

Specify the section group attribute value. The following are the attributes with their supported values:

xml_enable: Specify boolean values as TRUE or FALSE, T or F, YES or NO, Y or N, ON or OFF, or 1 or 0.

Related Topics

"CREATE_SECTION_GROUP"

8.53 SET SECTION ATTRIBUTE

Use <code>SET_SECTION_ATTRIBUTE</code> to specify attributes or properties for a given section.

The **attribute** names listed under "Syntax" are supported. Note that some attributes only apply to sections that are tokenized. The following section types are tokenized:

- Field sections
- Zone sections
- SDATA sections

Syntax

group_name

Specify the name of the section group.

section_name

Specify the name of the section.

attribute

Specify this attribute for SDATA sections:

Visible section attribute

This attribute works with FIELD sections only. For FIELD sections:

Specify TRUE to make the text visible within the rest of the document. By default, the visible flag is FALSE. This means that Oracle Text indexes the text within field sections as a sub-document separate from the rest of the document. However, you can set the visible flag to TRUE if you want text within the field section to be indexed as part of the enclosing document.

For field sections, *attribute* will override the value specified in CTX DDL.ADD FIELD SECTION.

An error is thrown if you try to set the visible attribute for a zone section.

An error is thrown if the visible attribute is set on a non-tokenized section.

- save_copy. Set to True or False. The save_copy option is valid for all types of sections, but only SDATA attributes are fetched from \$D table. The rest of the sections are stored for display purposes only (depending on value of save_copy). SDATA sections are never stored for display purposes, but are stored independently (in a separate column of \$D table) for efficient fetching (depending on value of save_copy). For all sections (except for SDATA sections): A section is either displayed or discarded during document service procedures (snippet, markup, highlight) depending on the value of save_copy.
- optimized for section attribute

This attribute makes an SDATA section optimal for search, optimal for sort, or optimal for both search and sort. These are achieved by setting the attribute value to search, sort, or sort and search.

- search provides efficient searching on SDATA sections.
- sort provides efficient sorting on SDATA sections. This is the default value.
- sort and search provides efficient searching and sorting on SDATA sections.

value

Specify the attribute value. Specify boolean values as TRUE or FALSE, T or F, YES or NO, Y or N, ON or OFF, or 1 or 0.

Example

The following example creates a basic section group called sg, adds a SDATA section to it and marks that SDATA section to be searchable by using the ctx ddl.set section attribute:

```
begin
    exec ctx_ddl.create_section_group('sg', 'basic_section_group');
    exec ctx_ddl.add_sdata_section('sg', 'sec1', 'sec1', 'varchar2');
    exec ctx_ddl.set_section_attribute('sg', 'sec1', 'optimized_for', 'search');
end;
```

Notes

Like CTX_DDL.SET_ATTRIBUTE, this procedure issues a commit.

Related Topic

See also the "Searching Document Sections in Oracle Text" chapter of *Oracle Text Application Developer's Guide*.

8.54 SYNC_INDEX

Synchronizes the index to process inserts, updates, and deletes to the base table.



Because CTX_DDL.SYNC_INDEX issues implicit commits, calling CTX_DDL.SYNC_INDEX in a trigger is strongly discouraged. Doing so can result in errors being raised, as both SYNC_INDEX and post-commit R LOB maintenance try to update the same R LOB.

Syntax

idx name

Specify the name of the index to synchronize.

Note:

When idx_name is null, all CONTEXT and CTXRULE indexes that have pending changes are synchronized. You must be connected as ctxsys to perform this operation. Each index or index partition is synchronized in sequence, one after the other. Because of this, the individual syncs are performed with locking set to NOWAIT and maxtime set to 0. Any values that you specify for locking or maxtime on the SYNC_INDEX call are ignored. However, the memory and parallel_degree parameters are passed on to the individual synchronizations.

memory

Specify the runtime memory to use for synchronization. This value overrides the DEFAULT INDEX MEMORY system parameter.

The memory parameter specifies the amount of memory Oracle Text uses for the synchronization operation before flushing the index to disk. Specifying a large amount of memory:

- Improves indexing performance because there is less I/O
- Improves guery performance and maintenance because there is less fragmentation
- The indexing memory size specified in the second argument applies to each parallel worker. For example, if the memory argument is set to 500M and parallel_degree is set to 2, then ensure that there is at least 1GB of memory available on the system used for the parallel SYNC INDEX.



Specifying smaller amounts of memory increases disk I/O and index fragmentation, but might be useful when runtime memory is scarce.

part name

If your index is a local index, then the <code>part_name</code> parameter is not mandatory. You may set <code>part_name</code> to specify the name of the index partition to synchronize, otherwise all index partitions are synchronized.

If your index is a global, nonpartitioned index, then specify NULL, which is the default.

parallel degree

Specify the degree to run parallel synchronize. A number greater than 1 turns on parallel synchronize. The actual degree of parallelism might be smaller depending on your resources.

maxtime

Indicate a suggested time limit on the operation, in minutes. SYNC_INDEX will process as many documents in the queue as possible within the time limit. The maxtime value of NULL is equivalent to CTX_DDL.MAXTIME_UNLIMITED. This parameter is ignored when SYNC_INDEX is invoked without an index name, in which case maxtime value of 0 is used instead. The locking parameter is ignored for automatic syncs (that is, SYNC ON COMMIT or SYNC EVERY). The time limit specified is treated as approximate. The actual time taken may be somewhat less than or greater than what you specify. The "time clock" for maxtime does not start until the SYNC lock is acquired.

locking

Configure how SYNC_INDEX deals with the situation where another sync is already running on the same index or index partition. When locking is ignored because SYNC_INDEX is invoked without an index name, then locking value of LOCK_NOWAIT is used instead. The locking parameter is ignored for automatic syncs (that is, SYNC ON COMMIT or SYNC EVERY). The options for locking are:

Locking Parameter	Description
CTX_DDL.LOCK_WAIT	If another sync is running, wait until the running sync is complete, then begin sync. (In the event of not being able to get a lock, it will wait forever and ignore the maxtime setting.)
CTX_DDL.LOCK_NOWAIT	If another sync is running, immediately returns without error.
CTX_DDL.LOCK_NOWAIT_ERROR	If another sync is running, error "DRG-51313: timeout while waiting for DML or optimize lock" is raised.

Example

The following example synchronizes the index myindex with 2 megabytes of memory:

```
begin
ctx_ddl.sync_index('myindex', '2M');
end;
```

The following example synchronizes the part1 index partition with 2 megabytes of memory:

```
begin
ctx ddl.sync index('myindex', '2M', 'part1');
```



end;

Notes

• For indexes with manual maintenance, this API launches sync in the foreground and returns after the running sync is complete.

For indexes with automatic maintenance, instead of running sync in the foreground, this API waits for any background sync event to finish and returns after the sync is complete. The background CTX DDL.SYNC INDEX operation performs the following steps in an order:

- 1. Resets all events (waiting for retry) for an index or index partition.
- 2. Waits for the background maintenance to finish, and performs SYNC-Mapping (Sync-M) in the foreground.
- 3. Posts the Scheduler process to start processing pending events in the background.
- 4. Waits for any background process to complete if the <code>locking</code> parameter is set to <code>CTX_DDL.LOCK_WAIT</code>. For all other <code>locking</code> parameter values, returns after completing <code>Sync-M</code>.

The values of the memory, parallel_degree, maxtime, and direct_path parameters are ignored.

If some background events are delayed or cannot complete, CTX_DDL.SYNC_INDEX returns ORA-30608 and logs an error message in the CTX_USER_BACKGROUND_EVENTS, CTX_BACKGROUND_EVENTS, and V\$TEXT_WAITING_EVENTS views.

For detailed information about these steps, see Oracle Text Application Developer's Guide.

- You can run CTX_DDL.SYNC_INDEX and CTX_DDL.OPTIMIZE_INDEX at the same time. You can also run CTX_DDL.SYNC_INDEX and CTX_DDL.OPTIMIZE_INDEX with parallelism at the same time. However, you should not run CTX_DDL.SYNC_INDEX with parallelism at the same time as CTX_DDL.OPTIMIZE_INDEX, nor CTX_DDL.SYNC_INDEX with parallelism at the same time as CTX_DDL.OPTIMIZE_INDEX with parallelism. If you should run one of these combinations, no error is generated; however, one operation will wait until the other is done.
- For indexes with a staging table (\$G), SYNC_INDEX automatically merges data back from the staging table to the permanent index table (\$I) when the threshold of rows inserted into the staging table exceeds the value of the STAGE_ITAB_MAX_ROWS setting and the STAGE_ITAB_MAX_ROWS is set to a value different than zero. The merge process uses a degree of parallelism of 4. Therefore, there is no need to explicitly run CTX_DDL.OPTIMIZE_INDEX in MERGE mode or to manually schedule a background job doing the same.

If you want to submit a user-owned DBMS_SCHEDULER background job to run CTX_DDL.OPTIMIZE_INDEX in MERGE mode, then you must explicitly set the STAGE_ITAB_MAX_ROWS attribute to 0. This turns off the automatic merge process that occurs during SYNC_INDEX. The user-submitted background job then periodically merges rows from \$G to \$I.

- You can set STAGE_ITAB_AUTO_OPT to enable automatic optimize merge. This setting automatically merges rows from \$G to \$I in the background.
 - When STAGE_ITAB_MAX_ROWS is set to a value greater than 0 and the automatic optimize merge is not enabled using STAGE_ITAB_AUTO_OPT, some SYNC operations may take an unexpectedly long time to complete due to the merging of rows from \$G to \$I.
- You cannot run DDL ALTER TABLE .. MODIFY concurrently with an index synchronization or index maintenance operation, such as SYNC_INDEX.



You cannot sync or optimize an index that is owned by a different schema. Doing so results
in the "DRG-10016: You must be the owner to modify this object" error.

Related Topics

ALTER INDEX

Use the ALTER INDEX statement to change or rebuild an existing index, such as Oracle Text index, Oracle Text search index, JSON search index, XML search index, or hybrid vector index.

8.55 UNSET_ATTRIBUTE

Removes a set attribute from a preference.

Syntax

preference_name

Specify the name of the preference.

attribute_name

Specify the name of the attribute.

Example

Enabling/Disabling Alternate Spelling

The following example shows how you can enable alternate spelling for German and disable alternate spelling with CTX_DDL.UNSET_ATTRIBUTE:

```
begin
ctx_ddl.create_preference('GERMAN_LEX', 'BASIC_LEXER');
ctx_ddl.set_attribute('GERMAN_LEX', 'ALTERNATE_SPELLING', 'GERMAN');
end;
```

To disable alternate spelling, use the CTX DDL.UNSET ATTRIBUTE procedure as follows:

```
begin
ctx_ddl.unset_attribute('GERMAN_LEX', 'ALTERNATE_SPELLING');
end;
```

Related Topics

"SET ATTRIBUTE"

8.56 UNSET_SEC_GRP_ATTR

Removes a section group-specific attribute.

Syntax

group_name

Specify the name of the section group.



attribute name

Specify the name of the attribute.

Related Topics

"UNSET ATTRIBUTE"

8.57 UPDATE SUB LEXER

Updates a sub-lexer and modifies its multi-lexer preference, language, or sub-lexer. You can also update default sub-lexers using this procedure. This procedure can be used in conjunction with the CTX DDL.PREFERENCE IMPLICIT COMMIT variable.



"PREFERENCE_IMPLICIT_COMMIT" for information about setting this variable

Syntax

lexer name

Specify the name of the multi-lexer preference that needs to be updated.

language

Specify the language name of the sub-lexer. Use <code>DEFAULT</code> for the default sub-lexers. See "language" for information on how to specify the globalization support language name or abbreviation of the sub-lexer.

sub lexer

Specify the name of the sub-lexer to use for this language.

Related Topics

```
"ADD_SUB_LEXER"

"REMOVE_SUB_LEXER"
```

8.58 UPDATE POLICY

Updates a policy created with CREATE_POLICY. Replaces the preferences of the policy. Null arguments are not replaced.

Syntax



```
stoplist IN VARCHAR2 DEFAULT NULL, wordlist IN VARCHAR2 DEFAULT NULL);
```

policy_name

Specify the name of the policy to update.

filter

Specify the filter preference to use.

section group

Specify the section group to use.

lexer

Specify the lexer preference to use.

stoplist

Specify the stoplist to use.

wordlist

Specify the wordlist to use.

8.59 UPDATE_SDATA

UPDATE_SDATA is an index API that modifies the specified SDATA values in the index. This API does not store or modify column values in a base table, where the base table column may have been used as an SDATA section.



The UPDATE_SDATA API in Oracle Text is deprecated in Oracle Database 23ai. Instead of modifying the index, Oracle recommends that you update the underlying data.

Export/import operations rebuild the index from the base table using the specified preferences. Since modifications made using the <code>UPDATE_SDATA</code> API are not present in the base table, the export/import operation does not preserve these changes.

UPDATE_SDATA modifies temporary metadata it adds in the index table, not the base table. It cannot be used to directly add metadata. For export/import of metadata that is persistent, create a base table column that contains the metadata values. You can then update the metadata through the column in the base table.

UPDATE SDATA truncates data which is larger than 249 bytes.

Syntax



idx_name

Specify the name of the index.

section_name

Specify the name of the SDATA section.

sdata_value

Specify the new SDATA value.

sdata_rowid

Specify the rowid for which the SDATA value needs to be updated.

part_name

Specify the name of the locally partitioned index, if applicable. Specify \mathtt{NULL} for the global index.

Related Topics

"SDATA"

"ADD_SDATA_COLUMN"

"ADD_SDATA_SECTION"



Chapter 8, "Searching Document Sections in Oracle Text" in *Oracle Text Application Developer's Guide*



CTX_DOC Package

The CTX_DOC PL/SQL package provides procedures and functions for requesting document services, such as highlighting extracted text or generating a list of themes for a document.

About CTX_DOC Package Procedures

The ${\tt CTX}\ {\tt DOC}$ package includes the following procedures and functions:

Name	Description	
FILTER	Generates a plain text or HTML version of a document.	
GIST	Generates a Gist or theme summaries for a document.	
HIGHLIGHT	Generates plain text or HTML highlighting offset information for a document.	
IFILTER	Generates a plain text version of binary data. Can be called from a USER_DATASTORE procedure.	
MARKUP	Generates a plain text or HTML version of a document with query terms highlighted.	
PKENCODE	Encodes a composite textkey string (value) for use in other CTX_DOC procedures.	
POLICY_FILTER	Generates a plain text or HTML version of a document, without requiring an index.	
POLICY_GIST	Generates a Gist or theme summaries for a document, without requiring an index.	
POLICY_HIGHLIGHT	Generates plain text or HTML highlighting offset information for a document, without requiring an index.	
POLICY_LANGUAGES	Provides the ability to fetch the language for a section of text.	
POLICY_MARKUP	Generates a plain text or HTML version of a document with query terms highlighted, without requiring an index.	
POLICY_NOUN_PHRASES	Extracts noun phrases for a document.	
POLICY_PART_OF_SPEECH	Extracts the part of speech for each word in a document.	
POLICY_SNIPPET	Generates a concordance for a document, based on query terms, without requiring an index.	
POLICY_STEMS	Extracts stems for each word in a body of text.	
POLICY_THEMES	Generates a list of themes for a document, without requiring an index.	
POLICY_TOKENS	Generates all index tokens for a document, without requiring an index.	
SENTIMENT	Performs sentiment analysis for a single document and provides a separate sentiment score for each segment within the document.	
SENTIMENT_AGGREGATE	Performs sentiment analysis for a single document and provides an aggregate sentiment score for the entire document.	
SET_KEY_TYPE	Sets CTX_DOC procedures to accept rowid or primary key document identifiers.	

Name	Description
SNIPPET	Generates a concordance for a document, based on query terms.
THEMES	Generates a list of themes for a document.
TOKENS	Generates all index tokens for a document.

The performance of the procedures SNIPPET, HIGHLIGHT, and MARKUP can be improved by using the forward index feature, and the performance of the procedures FILTER, GIST, THEMES. TOKENS can be improved by using the save copy feature of Oracle Text.



Oracle Text Application Developer's Guide for more information about forward index and save copy features

9.1 About CTX DOC Package Procedures

Many of the CTX_DOC PL/SQL package procedures exist in two versions: those that make use of indexes, and those that do not. Those that do not make use of indexes are called "policy-based" procedures. They are offered because there are times when you may want to use document services on a single document without creating a CONTEXT index in advance. Policy-based procedures enable you to do this.

The policy_* procedures mirror the conventional in-memory document services and are used with policy_name replacing index_ name, and document of type VARCHAR2, CLOB, BLOB, or BFILE replacing textkey. Thus, you need not create an index to obtain document services output with these procedures.

For the procedures that generate character offsets and lengths, such as HIGHLIGHT and TOKENS, Oracle Text follows USC-2 codepoint semantics.



The APIs in the CTX_DOC package do not support identifiers that are prefixed with the schema or the owner name.

9.2 FILTER

Use the $\mathtt{CTX_DOC.FILTER}$ procedure to generate either a plain text or HTML version of a document.

You can store the rendered document in either a result table or in memory. This procedure is generally called after a query, from which you identify the document to be filtered.



Note:

The resultant HTML document does not include graphics.

Syntax 1: In-memory Result Storage

Syntax 2: Result Table Storage

```
exec CTX_DOC.FILTER(

index_name IN VARCHAR2,

textkey IN VARCHAR2,

restab IN VARCHAR2,

query_id IN NUMBER DEFAULT 0,

plaintext IN BOOLEAN DEFAULT FALSE,

use saved copy IN NUMBER DEFAULT CTX DOC.SAVE COPY FALLBACK);
```

index_name

Specify the name of the index associated with the text column containing the document identified by textkey.

textkey

Specify the unique identifier (usually the primary key) for the document. The textkey parameter can be as follows:

- a single column primary key value
- encoded specification for a composite (multiple column) primary key. Use CTX_DOC.PKENCODE
- the rowid of the row containing the document

Toggle between primary key and rowid identification using CTX_DOC.SET_KEY_TYPE.

restab

You can specify that this procedure store the marked-up text to either a table or to an inmemory CLOB.

To store results to a table, specify the name of the table. The table to which you want to store results must exist before you make this call.



"Filter Table" in Oracle Text Result Tables for more information about the structure of the filter result table

To store results in memory, specify the name of the CLOB locator. If restab is NULL, then a temporary CLOB is allocated and returned. You must de-allocate the locator after using it with DBMS LOB.FREETEMPORARY().



If restab is not NULL, then the CLOB is truncated before the operation.

query_id

Specify an identifier to use to identify the row inserted into restab.

When $query_id$ is not specified or set to NULL, it defaults to 0. You must manually truncate the table specified in restab.

plaintext

Specify TRUE to generate a plaintext version of the document. Specify FALSE to generate an HTML version of the document if you are using the AUTO_FILTER filter or indexing HTML documents.

use saved copy

Specify whether to refer to the \$D table to fetch the copy of the document, and what action to take when the copy of the document is not available in the \$D table.

You can specify one of the following values for the use saved copy parameter:

- CTX_DOC.SAVE_COPY_FALLBACK: Fetch the copy of the document from the \$D table. If the
 copy of the document is not present in the \$D table, then fetch the document from the
 datastore.
- CTX_DOC.SAVE_COPY_ERROR: Fetch the copy of the document from the \$D table. If the SAVE_COPY option of the index is set to none and the document is not present in the \$D table, then show the CTX_DOC.SAVE_COPY_ERROR error. If the SAVE_COPY option of the index is set to plaintext or filtered and the document is not present in the \$D table, then show the nodoc_err error. If the document is present in the \$D table, then retrieve the document from the datastore.
- CTX DOC.SAVE COPY IGNORE: Always fetch the document from the datastore.

The default value is CTX DOC. SAVE COPY FALLBACK.

Example

In-Memory Filter

The following code shows how to filter a document to HTML in memory.

```
declare
mklob clob;
amt number := 40;
line varchar2(80);

begin
  ctx_doc.filter('myindex','1', mklob, FALSE);
  -- mklob is NULL when passed-in, so ctx-doc.filter will allocate a temporary
  -- CLOB for us and place the results there.
dbms_lob.read(mklob, amt, 1, line);
dbms_output.put_line('FIRST 40 CHARS ARE:'||line);
  -- have to de-allocate the temp lob
dbms_lob.freetemporary(mklob);
end;
```

Create the filter result table to store the filtered document as follows:

To obtain a plaintext version of document with textkey 20, enter the following statement:

```
begin
ctx_doc.filter('newsindex', '20', 'filtertab', '0', TRUE);
end:
```

9.3 GIST

Use the CTX_DOC.GIST procedure to generate gist and theme summaries for a document. You can generate paragraph-level or sentence-level gists or theme summaries.

Syntax 1: In-Memory Storage

Syntax 2: Result Table Storage

```
index_name IN VARCHAR2,
textkey IN VARCHAR2,
restab IN VARCHAR2,
query_id IN NUMBER DEFAULT 0,
glevel IN VARCHAR2 DEFAULT 'P',
pov IN VARCHAR2 DEFAULT NULL,
numParagraphs IN NUMBER DEFAULT 16,
maxPercent IN NUMBER DEFAULT 10,
num_themes IN NUMBER DEFAULT 50,
use_saved_copy IN NUMBER DEFAULT CTX_DOC.SAVE_COPY_FALLBACK);
```

index name

Specify the name of the index associated with the text column containing the document identified by textkey.

textkey

Specify the unique identifier (usually the primary key) for the document. The textkey parameter can be as follows:

- a single column primary key value
- an encoded specification for a composite (multiple column) primary key. To encode a composite textkey, use the CTX DOC.PKENCODE procedure
- the rowid of the row containing the document

Toggle between primary key and rowid identification using CTX DOC. SET KEY TYPE.

restab

Specify that this procedure store the gist and theme summaries to either a table or to an inmemory CLOB.

To store results to a table specify the name of an existing table.



"Gist Table" in Oracle Text Result Tables

To store results in memory, specify the name of the CLOB locator. If restab is NULL, then a temporary CLOB is allocated and returned. You must de-allocate the locator after using it.

If restab is not NULL, then the CLOB is truncated before the operation.

query_id

Specify an identifier to use to identify the row(s) inserted into restab.

glevel

Specify the type of gist or theme summary to produce. The possible values are:

- P for paragraph
- S for sentence

The default is P.

pov

Specify whether a gist or a single theme summary is generated. The type of gist or theme summary generated (sentence-level or paragraph-level) depends on the value specified for <code>glevel</code>.

To generate a gist for the entire document, specify a value of 'GENERIC' for pov. To generate a theme summary for a single theme in a document, specify the theme as the value for pov. When using result table storage, if you do not specify a value for pov, then this procedure returns the generic gist plus up to 50 theme summaries for the document. When using in-memory result storage to a CLOB, you must specify a pov. However, if you do not specify a pov, then this procedure generates only a generic gist for the document.



The pov parameter is case sensitive. To return a gist for a document, specify 'GENERIC' in all uppercase. To return a theme summary, specify the theme exactly as it is generated for the document.

Only the themes generated by THEMES for a document can be used as input for pov.

numParagraphs

Specify the maximum number of document paragraphs (or sentences) selected for the document gist or theme summaries. The default is 16.



Note:

The numParagraphs parameter is used only when this parameter yields a smaller gist or theme summary size than the gist or theme summary size yielded by the maxPercent parameter.

This means that the system always returns the smallest size gist or theme summary.

maxPercent

Specify the maximum number of document paragraphs (or sentences) selected for the document gist or theme summaries as a percentage of the total paragraphs (or sentences) in the document. The default is 10.

Note:

The maxPercent parameter is used only when this parameter yields a smaller gist or theme summary size than the gist or theme summary size yielded by the numParagraphs parameter.

This means that the system always returns the smallest size gist or theme summary.

num themes

Specify the number of theme summaries to produce when you do not specify a value for pov. For example, if you specify 10, this procedure returns the top 10 theme summaries. The default is 50.

If you specify 0 or NULL, then this procedure returns all themes in a document. If the document contains more than 50 themes, only the top 50 themes show conceptual hierarchy.

use_saved_copy

Specify whether to refer to the \$D table to fetch the copy of the document, and what action to take when the copy of the document is not available in the \$D table.

You can specify one of the following values for the use saved copy parameter:

- CTX_DOC.SAVE_COPY_FALLBACK: Fetch the copy of the document from the \$D table. If the copy of the document is not present in the \$D table, then fetch the document from the data store.
- CTX_DOC.SAVE_COPY_ERROR: Fetch the copy of the document from the \$D table. If the copy of the document is not present in the \$D table, then show an error message. Specify this value when you want to implement a specific fallback logic when the copy of the document is not available in the \$D table.
- CTX DOC.SAVE COPY IGNORE: Always fetch the document from the data store.

The default value is CTX DOC.SAVE COPY FALLBACK.

Examples

In-Memory Gist

The following example generates a non-default size generic gist of at most 10 paragraphs. The result is stored in memory in a CLOB locator. The code then de-allocates the returned CLOB locator after using it.

set serveroutput on;
declare
 gklob clob;



```
amt number := 40;
line varchar2(80);

begin
  ctx_doc.gist('newsindex','34',gklob, pov => 'GENERIC',numParagraphs => 10);
  -- gklob is NULL when passed-in, so ctx-doc.gist will allocate a temporary
  -- CLOB for us and place the results there.

dbms_lob.read(gklob, amt, 1, line);
  dbms_output.put_line('FIRST 40 CHARS ARE:'||line);
  -- have to de-allocate the temp lob
  dbms_lob.freetemporary(gklob);
end;
```

Result Table Gists

The following example creates a gist table called CTX GIST:

Gists and Theme Summaries

The following example returns a default sized paragraph-level gist for document 34 as well as the top 10 theme summaries in the document:

```
begin
   ctx_doc.gist('newsindex','34','CTX_GIST', 1, num_themes=>10);
end;
```

The following example generates a non-default size gist of at most 10 paragraphs:

```
begin
  ctx_doc.gist('newsindex','34','CTX_GIST',1,pov =>'GENERIC',numParagraphs=>10);
end:
```

The following example generates a gist whose number of paragraphs is at most 10 percent of the total paragraphs in document:

```
begin
   ctx_doc.gist('newsindex','34','CTX_GIST',1,pov => 'GENERIC', maxPercent => 10);
end;
```

Theme Summary

The following example returns a paragraph-level theme summary for *insects* for document 34. The default theme summary size is returned.

```
begin
   ctx_doc.gist('newsindex','34','CTX_GIST',1, pov => 'insects');
end;
```

9.4 HIGHLIGHT

Use the CTX_DOC.HIGHLIGHT procedure to generate highlight offsets for a document. The offset information is generated for the terms in the document that satisfy the query you specify. These highlighted terms are either the words that satisfy a word query or the themes that satisfy an ABOUT query.



You can generate highlight offsets for either plaintext or HTML versions of the document. The table returned by $CTX_DOC.HIGHLIGHT$ does not include any graphics found in the original document. Apply the offset information to the same documents filtered with $CTX_DOC.FILTER$.

You usually call this procedure after a query, from which you identify the document to be processed. You can store the highlight offsets to either an in-memory PL/SQL table or a result table.

Note that for queries that have predicates used mainly for filtering documents at query time, the predicates are ignored during highlighting. This applies to SNIPPET, MARKUP and HIGHLIGHT procedures. The following predicates are treated as filter predicates for this purpose: SDATA, HASPATH, and WITHIN/INPATH searching inside XML attributes.

See CTX_DOC.POLICY_HIGHLIGHT for a version of this procedure that does not require an index.

The performance of the procedures SNIPPET, HIGHLIGHT, and MARKUP can be improved by using the forward index feature of Oracle Text.



Oracle Text Application Developer's Guide for more information about forward index

Syntax 1: In-Memory Result Storage

```
exec CTX_DOC.HIGHLIGHT(
    index_name IN VARCHAR2,
    textkey IN VARCHAR2,
    text_query IN VARCHAR2,
    restab IN OUT NOCOPY HIGHLIGHT_TAB,
    plaintext IN BOOLEAN DEFAULT FALSE,
    use_saved_copy IN NUMBER DEFAULT CTX_DOC.SAVE_COPY_FALLBACK);

exec CTX_DOC.HIGHLIGHT_CLOB_QUERY(
    index_name IN VARCHAR2,
    textkey IN VARCHAR2,
    text_query IN CLOB,
    restab IN OUT NOCOPY HIGHLIGHT_TAB,
    plaintext IN BOOLEAN DEFAULT FALSE,
    use saved copy IN NUMBER DEFAULT CTX DOC.SAVE COPY FALLBACK);
```

Syntax 2: Result Table Storage



index name

Specify the name of the index associated with the text column containing the document identified by textkey.

textkey

Specify the unique identifier (usually the primary key) for the document.

The textkey parameter can be as follows:

- a single column primary key value
- encoded specification for a composite (multiple column) primary key. Use the CTX_DOC.PKENCODE procedure.
- the rowid of the row containing the document

Toggle between primary key and rowid identification using CTX DOC.SET KEY TYPE.

text_query

Specify the original query expression used to retrieve the document. If NULL, no highlights are generated.

If text_query includes wildcards, stemming, fuzzy matching which result in stopwords being returned, <code>HIGHLIGHT</code> does not highlight the stopwords.

If text_query contains the threshold operator, the operator is ignored. The HIGHLIGHT procedure always returns highlight information for the entire result set.

restab

You can specify that this procedure store highlight offsets to either a table or to an in-memory PL/SQL table.

To store results to a table specify the name of the table. The table must exist before you call this procedure.

See Also:

"Highlight Table" in Oracle Text Result Tables for more information about the structure of the highlight result table.

To store results to an in-memory table, specify the name of the in-memory table of type CTX DOC.HIGHLIGHT TAB. The HIGHLIGHT TAB datatype is defined as follows:

```
type highlight_rec is record (
  offset number,
  length number
);
type highlight_tab is table of highlight_rec index by binary_integer;
CTX DOC.HIGHLIGHT clears HIGHLIGHT TAB before the operation.
```

query_id

Specify the identifier used to identify the row inserted into restab. When query_id is not specified or set to NULL, it defaults to 0. You must manually truncate the table specified in restab.



plaintext

Specify TRUE to generate a plaintext offsets of the document. Specify FALSE to generate HTML offsets of the document if you are using the AUTO FILTER filter or indexing HTML documents.

use_saved_copy

Specify whether to refer to the \$D table to fetch the copy of the document, and what action to take when the copy of the document is not available in the \$D table. The default value is CTX DOC.SAVE COPY FALLBACK.

You can specify one of the following values for the use saved copy parameter:

- CTX_DOC.SAVE_COPY_FALLBACK: Fetch the copy of the document from the \$D table. If the copy of the document is not present in the \$D table, then fetch the document from the data store.
- CTX_DOC.SAVE_COPY_ERROR: Fetch the copy of the document from the \$D table. If the copy of the document is not present in the \$D table, then show an error message. Specify this value when you want to implement a specific fallback logic when the copy of the document is not available in the \$D table.
- CTX DOC.SAVE COPY IGNORE: Always fetch the document from the data store.

Examples

Create Highlight Table

Create the highlight table to store the highlight offset information:

Word Highlighting in the Presence of Filters

When performing highlight on queries such as the following, only the keyword ("dog" in these examples) will be highlighted. The filtering predicates after the AND operator will be ignored.

```
begin
ctx_doc.highlight('newsindex', '20', 'dog AND cat WITHIN titlesection@name', 'hightab',
0, FALSE);
end;
begin
ctx_doc.highlight('newsindex', '20', 'dog AND SDATA(price > 100)', 'hightab', 0, FALSE);
end;
```

Word Highlight Offsets

To obtain HTML highlight offset information for document 20 for the word *dog*:

```
begin
ctx_doc.highlight('newsindex', '20', 'dog', 'hightab', 0, FALSE);
end;

begin
ctx_doc.highlight('newsindex', '20', 'dog AND cat WITHIN titlesection', 'hightab', 0,
FALSE);
end;
```

Theme Highlight Offsets

Assuming the index *newsindex* has a theme component, obtain HTML highlight offset information for the theme query of *politics* by issuing the following guery:



```
begin
ctx_doc.highlight('newsindex', '20', 'about(politics)', 'hightab', 0, FALSE);
end;
```

The output for this statement are the offsets to highlighted words and phrases that represent the theme of *politics* in the document.

Restrictions

CTX_DOC.HIGHLIGHT does not support the use of query templates or highlighting XML attribute values.

Related Topics

```
"POLICY_HIGHLIGHT"
"MARKUP"
"SNIPPET"
```

9.5 IFILTER

Use this procedure to filter binary data to text.

This procedure takes binary data (BLOB IN), filters the data with the AUTO_FILTER filter, and writes the text version to a CLOB. (Any graphics in the original document are ignored.) CTX_DOC.IFILTER employs the safe callout, and it does not require an index, as CTX_DOC.FILTER does.



This procedure will not be supported in future releases. Applications should use CTX_DOC.POLICY_FILTER instead.

Requirements

Because CTX_DOC.IFILTER employs the safe callout mechanism, the SQL*Net listener must be running and configured for extproc agent startup.

Syntax

```
CTX_DOC.IFILTER(data IN BLOB, text IN OUT NOCOPY CLOB);
```

data

Specify the binary data to be filtered.

text

Specify the destination CLOB. The filtered data is placed in here. This parameter must be a valid CLOB locator that is writable. Passing NULL or a non-writable CLOB will result in an error. Filtered text will be appended to the end of existing content, if any.

Example

The document text used in a MATCHES query can be VARCHAR2 or CLOB. It does not accept BLOB input, so you cannot match filtered documents directly. Instead, you must filter the binary



content to CLOB using the AUTO_FILTER filter. Assuming the document data is in bind variable :doc blob:

```
declare
  doc_text clob;
begin
  -- create a temporary CLOB to hold the document text
  dbms_lob.createtemporary(doc_text, TRUE, DBMS_LOB.SESSION);
  -- call ctx_doc.ifilter to filter the BLOB to CLOB data
  ctx_doc.ifilter(:doc_blob, doc_text);
  -- now do the matches query using the CLOB version
  for c1 in (select * from queries where matches(query_string, doc_text)>0)
  loop
   -- do what you need to do here
  end loop;
  dbms_lob.freetemporary(doc_text);
end;
```

9.6 MARKUP

The CTX_DOC.MARKUP procedure takes a query specification and a document textkey and returns a version of the document in which the query terms are marked up. These marked-up terms are either the words that satisfy a word query or the themes that satisfy an ABOUT query.

You can set the marked-up output to be either plaintext or HTML. The marked-up document returned by CTX DOC.MARKUP does not include any graphics found in the original document.

You can use one of the predefined tag sets for marking highlighted terms, including a tag sequence that enables HTML navigation.

You usually call $CTX_DOC.MARKUP$ after a query, from which you identify the document to be processed.

You can store the marked-up document either in memory or in a result table.

Note that for queries that have predicates used mainly for filtering documents at query time, the predicates are ignored during MARKUP. The following predicates are treated as filter predicates for this purpose: SDATA, HASPATH, and WITHIN/INPATH searching inside XML attributes.

See CTX_DOC.POLICY_MARKUP for a version of this procedure that does not require an index.

The performance of the procedures SNIPPET, HIGHLIGHT, and MARKUP can be improved by using the forward index feature of Oracle Text.



Oracle Text Application Developer's Guide for more information about forward index



Note:

Oracle Text does not guarantee well-formed output from CTX.DOC.MARKUP, especially for terms that are already marked up with HTML or XML. In particular, unexpected nesting of markup tags may occasionally result.

Syntax 1: In-Memory Result Storage

Syntax 2: Result Table Storage

```
exec CTX DOC.MARKUP(
                IN VARCHAR2,
index name
textkey IN VARCHAR2,
text_query IN VARCHAR2,
restab IN VARCHAR2,
query_id IN NUMBER DEFAULT 0,
plaintext IN BOOLEAN DEFAULT FALSE,
tagset IN VARCHAR2 DEFAULT 'TEXT_DEFAULT', starttag IN VARCHAR2 DEFAULT NULL, endtag IN VARCHAR2 DEFAULT NULL,
prevtag
prevtag IN VARCHAR2 DEFAULT NULL, nexttag IN VARCHAR2 DEFAULT NULL,
use saved copy IN NUMBER DEFAULT CTX DOC. SAVE COPY FALLBACK);
exec CTX DOC.MARKUP CLOB QUERY(
index name IN VARCHAR2,
                  IN CLOB,
textkey
text_query IN VARCHAR2,
restab IN VARCHAR2,
query_id IN NUMBER DEFAULT 0,
plaintext IN BOOLEAN DEFAULT FALSE,
```



```
tagset IN VARCHAR2 DEFAULT 'TEXT_DEFAULT',
starttag IN VARCHAR2 DEFAULT NULL,
endtag IN VARCHAR2 DEFAULT NULL,
prevtag IN VARCHAR2 DEFAULT NULL,
nexttag IN VARCHAR2 DEFAULT NULL,
use_saved_copy IN NUMBER DEFAULT CTX_DOC.SAVE_COPY_FALLBACK);
```

index name

Specify the name of the index associated with the text column containing the document identified by textkey.

textkey

Specify the unique identifier (usually the primary key) for the document.

The textkey parameter can be as follows:

- A single column primary key value
- Encoded specification for a composite (multiple column) primary key. Use the CTX_DOC.PKENCODE procedure.
- The rowid of the row containing the document

Toggle between primary key and rowid identification using CTX_DOC.SET_KEY_TYPE.

text_query

Specify the original query expression used to retrieve the document.

If text_query includes wildcards, stemming, fuzzy matching which result in stopwords being returned, MARKUP does not highlight the stopwords.

If text_query contains the threshold operator, the operator is ignored. The MARKUP procedure always returns highlight information for the entire result set.

restab

You can specify that this procedure store the marked-up text to either a table or to an inmemory CLOB.

To store results to a table specify the name of the table. The result table must exist before you call this procedure.



For more information about the structure of the markup result table, see "Markup Table" in Oracle Text Result Tables.

To store results in memory, specify the name of the CLOB locator. If restab is NULL, a temporary CLOB is allocated and returned. You must de-allocate the locator after using it.

If restab is not NULL, the CLOB is truncated before the operation.

query_id

Specify the identifier used to identify the row inserted into restab.

When <code>query_id</code> is not specified or set to <code>NULL</code>, it defaults to 0. You must manually truncate the table specified in <code>restab</code>.

plaintext

Specify TRUE to generate plaintext marked-up document. Specify FALSE to generate a marked-up HTML version of document if you are using the AUTO_FILTER filter or indexing HTML documents.



tagset

Specify one of the following predefined tag sets. The second and third columns show how the different tags are defined for each tagset:

Tagset	Tag	Tag Value
TEXT_DEFAULT	starttag	<<<
TEXT_DEFAULT	endtag	>>>
HTML_DEFAULT	starttag	
HTML_DEFAULT	endtag	
HTML_NAVIGATE	starttag	
HTML_NAVIGATE	endtag	
HTML_NAVIGATE	prevtag	<pre><</pre>
HTML_NAVIGATE	nexttag	>

starttag

Specify the character(s) inserted by MARKUP to indicate the start of a highlighted term. The sequence of starttag, endtag, prevtag and nexttag with respect to the highlighted word is as follows:

... prevtag starttag word endtag nexttag...

endtag

Specify the character(s) inserted by MARKUP to indicate the end of a highlighted term.

prevtag

Specify the markup sequence that defines the tag that navigates the user to the previous highlight.

In the markup sequences prevtag and nexttag, you can specify the following offset variables which are set dynamically:

Offset Variable	Value
%CURNUM	the current offset number
%PREVNUM	the previous offset number
%NEXTNUM	the next offset number

See the description of the HTML NAVIGATE "tagset" for an example.

nexttag

Specify the markup sequence that defines the tag that navigates the user to the next highlight tag.

Within the markup sequence, you can use the same offset variables you use for prevtag. See the explanation for "prevtag" and the HTML NAVIGATE "tagset" for an example.

use_saved_copy

Specify whether to refer to the \$D table to fetch the copy of the document, and what action to take when the copy of the document is not available in the \$D table.

You can specify one of the following values for the use saved copy parameter:

- CTX_DOC.SAVE_COPY_FALLBACK: Fetch the copy of the document from the \$D table. If the copy of the document is not present in the \$D table, then fetch the document from the data store.
- CTX_DOC.SAVE_COPY_ERROR: Fetch the copy of the document from the \$D table. If the copy of the document is not present in the \$D table, then show an error message. Specify this



value when you want to implement a specific fallback logic when the copy of the document is not available in the \$D table.

CTX DOC.SAVE COPY IGNORE: Always fetch the document from the data store.

The default value is CTX_DOC.SAVE_COPY_FALLBACK.

Examples

In-Memory Markup

The following code takes document (the dog chases the cat), performs the assigned markup on it, and stores the result in memory.

```
set serveroutput on
drop table mark tab;
create table mark tab (id number primary key, text varchar2(80));
insert into mark tab values ('1', 'The dog chases the cat.');
create index mark tab idx on mark tab(text)
        indextype is ctxsys.context parameters
        ('filter ctxsys.null filter');
declare
mklob clob;
amt number := 40;
line varchar2(80);
begin
ctx doc.markup('mark tab idx','1','dog AND cat', mklob);
 -- mklob is NULL when passed-in, so ctx doc.markup will
 -- allocate a temporary CLOB for us and place the results there.
 dbms lob.read(mklob, amt, 1, line);
 dbms_output.put_line('FIRST 40 CHARS ARE:'||line);
 -- have to de-allocate the temp lob
 dbms lob.freetemporary(mklob);
 end;
```

The output from this example shows what the marked-up document looks like:

```
FIRST 40 CHARS ARE: The <<<dog>>> chases the <<<cat>>>.
```

Markup Table

Create the highlight markup table to store the marked-up document as follows:

Word Highlighting in HTML

You can also store your MARKUP results in a table. To create HTML highlight markup for the words *dog* or *cat* for document 23, enter the following examples:



Word Highlighting in the Presence of Filters

When performing markup on queries such as the following, only the keyword ("dog" in these examples) will be marked up. The filtering predicates after the AND operator will be ignored.

```
begin
  ctx doc.markup(index name => 'my index',
                       textkey \Rightarrow '23',
                       text query => 'dog AND cat WITHIN titlesection@name',
                       restab => 'markuptab',
                       query id => '1',
                       tagset => 'HTML DEFAULT');
end;
begin
  ctx doc.markup(index name => 'my index',
                       textkey \Rightarrow '23',
                       text_query => 'dog AND SDATA(price > 100)',
                       restab => 'markuptab',
                       query id => '1',
                       tagset => 'HTML_DEFAULT');
end;
```

Theme Highlighting in HTML

To create HTML highlight markup for the theme of *politics* for document 23, enter the following statement:

Restrictions

CTX_DOC.MARKUP does not support the use of query templates.

Related Topics

```
"POLICY_MARKUP"
```

"SNIPPET"



9.7 PKENCODE

The CTX_DOC.PKENCODE function converts a composite textkey list into a single string and returns the string.

The string created by PKENCODE can be used as the primary key parameter textkey in other CTX DOC procedures, such as CTX_DOC.THEMES and CTX_DOC.GIST.

Syntax

```
CTX DOC.PKENCODE (
        pk1 IN VARCHAR2,
        pk2 IN VARCHAR2 DEFAULT NULL,
        pk4 IN VARCHAR2 DEFAULT NULL,
        pk5 IN VARCHAR2 DEFAULT NULL,
        pk6 IN VARCHAR2 DEFAULT NULL,
        pk7 IN VARCHAR2 DEFAULT NULL,
        pk8 IN VARCHAR2 DEFAULT NULL,
        pk9 IN VARCHAR2 DEFAULT NULL,
        pk10 IN VARCHAR2 DEFAULT NULL,
        pk11 IN VARCHAR2 DEFAULT NULL,
        pk12 IN VARCHAR2 DEFAULT NULL,
        pk13 IN VARCHAR2 DEFAULT NULL,
        pk14 IN VARCHAR2 DEFAULT NULL,
        pk15 IN VARCHAR2 DEFAULT NULL,
        pk16 IN VARCHAR2 DEFAULT NULL)
RETURN VARCHAR2;
```

pk1-pk16

Each PK argument specifies a column element in the composite textkey list. You can encode at most 16 column elements.

Returns

String that represents the encoded value of the composite textkey.

Example

```
begin
ctx_doc.gist('newsindex',CTX_DOC.PKENCODE('smith', 14), 'CTX_GIST');
end;
```

In this example, *smith* and *14* constitute the composite textkey value for the document.

9.8 POLICY FILTER

Generates a plain text or an HTML version of a document. With this procedure, no CONTEXT index is required.

This procedure uses a trusted callout.

Syntax

```
ctx_doc.policy_filter(policy_name in VARCHAR2, document in [VARCHAR2|CLOB|BLOB|BFILE], restab in out nocopy CLOB, plaintext in BOOLEAN default FALSE, language in VARCHAR2 default NULL,
```



```
format in VARCHAR2 default NULL, charset in VARCHAR2 default NULL);
```

policy_name

Specify the policy name created with CTX DDL.CREATE POLICY.

document

Specify the document to filter.

restab

Specify the name of the CLOB locator.

plaintext

Specify TRUE to generate a plaintext version of the document. Specify FALSE to generate an HTML version of the document if you are using the AUTO_FILTER filter or indexing HTML documents.

language

Specify the language of the document. Use an Oracle Text supported language value as you would in the language column of the base table. See BASIC_LEXER in Oracle Text Indexing Elements.

format

Specify the format of the document. Use an Oracle Text supported format value, either TEXT, BINARY or IGNORE as you would specify in the format column of the base table. For more information, see the format column description in CREATE INDEX in Oracle Text SQL Statements and Operators .

charset

Specify the character set of the document. Use an Oracle Text supported value as you would specify in the charset column of the base table. See "Filter Types".

9.9 POLICY_GIST

Generates a gist or theme summary for document. You can generate paragraph-level or sentence-level gists or theme summaries. With this procedure, no CONTEXT index is required.

Syntax

policy_name

Specify the policy name created with CTX_DDL.CREATE_POLICY.

document

Specify the document for which to generate the Gist or theme summary.

restab

Specify the name of the CLOB locator.

glevel

Specify the type of gist or theme summary to produce. The possible values are:

- P for paragraph
- S for sentence

The default is P.

pov

Specify whether a gist or a single theme summary is generated. The type of gist or theme summary generated (sentence-level or paragraph-level) depends on the value specified for glevel.

To generate a gist for the entire document, specify a value of 'GENERIC' for pov. To generate a theme summary for a single theme in a document, specify the theme as the value for pov. When using result table storage and you do not specify a value for pov, this procedure returns the generic gist plus up to 50 theme summaries for the document.



The pov parameter is case sensitive. To return a gist for a document, specify 'GENERIC' in all uppercase. To return a theme summary, specify the theme *exactly* as it is generated for the document.

Only the themes generated by THEMES for a document can be used as input for pov.

numParagraphs

Specify the maximum number of document paragraphs (or sentences) selected for the document gist or theme summaries. The default is 16.



The numParagraphs parameter is used only when this parameter yields a smaller gist or theme summary size than the gist or theme summary size yielded by the maxPercent parameter.

This means that the system always returns the smallest size gist or theme summary.

maxPercent

Specify the maximum number of document paragraphs (or sentences) selected for the document gist or theme summaries as a percentage of the total paragraphs (or sentences) in the document. The default is 10.



Note:

The maxPercent parameter is used only when this parameter yields a smaller gist or theme summary size than the gist or theme summary size yielded by the numParagraphs parameter.

This means that the system always returns the smallest size gist or theme summary.

num themes

Specify the number of theme summaries to produce when you do not specify a value for pov. For example, if you specify 10, this procedure returns the top 10 theme summaries. The default is 50.

If you specify 0 or NULL, this procedure returns all themes in a document. If the document contains more than 50 themes, only the top 50 themes show conceptual hierarchy.

language

Specify the language of the document. Use an Oracle Text supported language value as you would in the language column of the base table. See "MULTI LEXER".

format

Specify the format of the document. Use an Oracle Text supported format value, either TEXT, BINARY or IGNORE as you would specify in the format column of the base table. For more information, see the format column description in "CREATE INDEX".

charset

Specify the character set of the document. Use an Oracle Text supported value as you would specify in the charset column of the base table.

9.10 POLICY HIGHLIGHT

Generates plain text or HTML highlighting offset information for a document. With this procedure, no CONTEXT index is required.

The offset information is generated for the terms in the document that satisfy the query you specify. These highlighted terms are either the words that satisfy a word query or the themes that satisfy an ABOUT query.

You can generate highlight offsets for either plaintext or HTML versions of the document. You can apply the offset information to the same documents filtered with CTX_DOC.FILTER.

Syntax



```
restab in out nocopy highlight_tab,
plaintext in boolean FALSE
language in VARCHAR2 default NULL,
format in VARCHAR2 default NULL,
charset in VARCHAR2 default NULL
```

policy name

);

Specify the policy name created with CTX_DDL.CREATE_POLICY.

document

Specify the document to generate highlighting offset information.

text_query

Specify the original query expression used to retrieve the document. If NULL, no highlights are generated.

If text_query includes wildcards, stemming, or fuzzy matching which result in stopwords being returned, this procedure does not highlight the stopwords.

If text_query contains the threshold operator, the operator is ignored. This procedure always returns highlight information for the entire result set.

restab

Specify the name of the highlight tab PL/SQL index-by-table type.



"HIGHLIGHT" for more information about the structure of the $highlight_tab$ table type

plaintext

Specify TRUE to generate a plaintext offsets of the document.

Specify FALSE to generate HTML offsets of the document if you are using the AUTO_FILTER filter or indexing HTML documents.

language

Specify the language of the document. Use an Oracle Text supported language value as you would in the language column of the base table. See "MULTI_LEXER" in Oracle Text Indexing Elements.

format

Specify the format of the document. Use an Oracle Text supported format value, either TEXT, BINARY or IGNORE as you would specify in the format column of the base table. For more information, see the format column description under "CREATE INDEX".

charset

Specify the character set of the document. Use an Oracle Text supported value as you would specify in the charset column of the base table.

Restrictions

CTX DOC.POLICY HIGHLIGHT does not support the use of query templates.



9.11 POLICY_LANGUAGES

Provides the ability to fetch the language for a section of text.

Returns a table of language descriptors and scores, where the score is the confidence level with which the system can assert that the supplied text is in the specific language.

Syntax

policy_name

A policy that was previously created using the CTX_DDL.CREATE_POLICY method. If the specified policy includes a sectioning preference, the API will honor the sectioning preference. For instance, if HTML sectioning is specified, then HTML tags will be removed before processing the input document.

document

A body of text for which the languages are to be extracted. The text is assumed to be plain text with UTF-8 character encoding.

restab

The result of the language extraction process. The result is a table of records. Each record has two attributes: the language string, and the score for each language string. The score can range from 0 to 100 and represents the confidence with which the system can assert that the supplied text is in the specified language. The resulting languages are returned in sorted order with the language with the most confidence appearing first.

The table layout for restab is similar to that for HIGHLIGHT.

Supported Languages for CTX_DOC.POLICY_LANGUAGES and POLICY_STEMS

You can use language extraction for text in all languages that are supported for AUTO LEXER.

Related Topics

AUTO LEXER Language Support

At index time, AUTO_LEXER automatically detects the language of the document, and tokenizes and stems the document appropriately.

- HIGHLIGHT
- CREATE POLICY

Creates a policy to use with the $CTX_DOC.POLICY_*$ procedures, certain Oracle Data Mining procedures, and the in-memory Text index.

POLICY_STEMS

9.12 POLICY MARKUP

Generates plain text or HTML version of a document with query terms highlighted. With this procedure, no CONTEXT index is required.

The CTX_DOC.POLICY_MARKUP procedure takes a query specification and a document and returns a version of the document in which the query terms are marked up. These marked-up terms are either the words that satisfy a word query or the themes that satisfy an ABOUT query.

You can set the marked-up output to be either plaintext or HTML.

You can use one of the predefined tag sets for marking highlighted terms, including a tag sequence that enables HTML navigation.

Syntax

policy_name

Specify the policy name created with CTX DDL.CREATE POLICY.

document

Specify the document to generate highlighting offset information.

text query

Specify the original query expression used to retrieve the document.

If text query includes a NULL, then this procedure will fail and generate errors.

If text_query includes wildcards, stemming, or fuzzy matching which result in stopwords being returned, then this procedure does not highlight the stopwords.

If text_query contains the threshold operator, the operator is ignored. This procedure always returns highlight information for the entire result set.

restab

Specify the name of the CLOB locator.



plaintext

Specify TRUE to generate a plaintext marked-up document. Specify FALSE to generate a marked-up HTML version of the document if you are using the AUTO_FILTER filter or indexing HTML documents.

tagset

Specify one of the following predefined tag sets. The second and third columns show how the different tags are defined for each tagset:

Tagset	Tag	Tag Value
TEXT_DEFAULT	starttag	<<<
TEXT_DEFAULT	endtag	>>>
HTML_DEFAULT	starttag	
HTML_DEFAULT	endtag	
HTML_NAVIGATE	starttag	
HTML_NAVIGATE	endtag	
HTML_NAVIGATE	prevtag	<
HTML_NAVIGATE	nexttag	>

starttag

Specify the character(s) inserted by MARKUP to indicate the start of a highlighted term. The sequence of starttag, endtag, prevtag and nexttag with regard to the highlighted word is as follows:

... prevtag starttag word endtag nexttag...

endtag

Specify the character(s) inserted by MARKUP to indicate the end of a highlighted term.

prevtag

Specify the markup sequence that defines the tag that navigates the user to the previous highlight.

In the markup sequences prevtag and nexttag, you can specify the following offset variables which are set dynamically:

Offset Variable	Value
%CURNUM	the current offset number
%PREVNUM	the previous offset number
%NEXTNUM	the next offset number

See the description of the HTML NAVIGATE tagset for an example "tagset".

nexttag

Specify the markup sequence that defines the tag that navigates the user to the next highlight tag.

Within the markup sequence, you can use the same offset variables you use for prevtag. See the explanation for prevtag and the HTML NAVIGATE "tagset" for an example.

language

Specify the language of the document. Use an Oracle Text supported language value as you would in the language column of the base table. See "MULTI_LEXER" in Oracle Text Indexing Elements.



format

Specify the format of the document. Use an Oracle Text supported format value, either TEXT, BINARY or IGNORE as you would specify in the format column of the base table. For more information, see the format column description in "CREATE INDEX".

charset

Specify the character set of the document. Use an Oracle Text supported value as you would specify in the charset column of the base table. See "Filter Types".

Restrictions

CTX DOC.POLICY MARKUP does not support the use of query templates.

9.13 POLICY_NOUN_PHRASES

Provides the ability to extract the noun phrases along with part-of-speech information for each word in each noun phrase from a given document.

For example, consider the following sentence:

"The mayor of Chicago is giving a brief press conference."

The noun phrases for this input are "mayor of Chicago" and "brief press conference." The subgroups in the input text are not returned. For instance, in the above example, subgroups such as "mayor, Chicago, brief press, press conference, press, conference" are not returned.

```
All AUTO_LEXER languages are supported for POLICY_NOUN_PHRASES and POLICY PART OF SPEECHPOLICY NOUN PHRASES.
```

Syntax

```
ctx_doc.policy_noun_phrases (
  policy_name in varchar2,
  document in varchar2 | CLOB,
  restab in out nocopy noun_phrase_tab,
  language in varchar2 default NULL,
  format in varchar2 default NULL,
  charset in varchar2 default NULL);
```

policy_name

Specify the policy name created with CTX_DDL.CREATE_POLICY.

document

A body of text for which the languages are to be extracted. The text is assumed to be plain text with UTF-8 character encoding.

restab

Specify the name of the CLOB locator.

language

Specify the language. See the list of supported languages in this section. If this parameter is null, the language will be automatically detected. There is a cost associated with language detection.

format

The format of the input text.

charset

The character set of the input text.

Abbreviations for Use with POLICY_NOUN_PHRASES and POLICY_PART_OF_SPEECH

This is a list of abbreviations that you can use in queries for POLICY_NOUN_PHRASES and POLICY_PART_OF_SPEECH. The examples use these abbreviations.

Table 9-1 Part of Speech Abbreviations

Abbreviation	Part of Speech
N	noun
propN	nounProper
V	verb
Adj	adjective
Adv	adverb
Prep	preposition
Part	particle
Punct	punct
Pro	pronoun
Wh	interrog
Det	determiner
Conj	conjunction
Card	numCardinal
Ord	numOrdinal
Suf	suffix
Pre	prefix
Acr	nounAcronym
Poss	poss
Unk	unknown

Example for POLICY_NOUN_PHRASES

The example in this section uses the abbreviations shown in the preceding table.

```
set serverout on
create or replace function toString(b boolean) return varchar2 is
   begin
    if (b) then
       return 'TRUE';
   end if;
   return 'FALSE';
end;
/
declare
  the_nps ctx_doc.noun_phrase_tab;
begin
  ctx_ddl.create_preference('rvlex', 'AUTO_LEXER');
  ctx_ddl.set_attribute('rvlex','mixed_case','YES');
  ctx_ddl.set_attribute('rvlex','timeout',0);
```

```
ctx ddl.create policy(policy name => 'rv policy 21',lexer => 'rvlex');
 ctx_doc.policy_noun_phrases('rv_policy_21','The mayor of Chicago is giving a
 brief press conference', the nps);
 dbms_output.put_line(the_nps.count);
 for i in 1..the nps.count loop
      if (the nps(i).is phrase start) then
        if (i>1) then
          dbms output.put(']');
          dbms output.new line;
        end if;
        dbms_output.put('Phrase{term, POS, is_in_lex, offset, len, is_phrase_
        start}:[');
      else
        dbms output.put(',');
      end if;
      dbms output.put('{' || the nps(i).term || ',' || the nps(i).pos tag || ','
      || toString(the nps(i).is in lexicon) || ',' || the nps(i).offset
      || ',' || the nps(i).length || ',' || toString(the nps(i).is phrase start)
      || '}');
      end loop;
      dbms output.put(']');
      dbms output.new_line;
end;
```

Output for this example:

```
Phrase{term, POS, is_in_lex, offset, len, is_phrase_start}:
[{The, Det, TRUE, 1, 3, TRUE}, {mayor, N, TRUE, 5, 5, FALSE},
{of, Prep, TRUE, 11, 2, FALSE}, {Chicago, propN, TRUE, 14, 7, FALSE}

Phrase{term, POS, is_in_lex, offset, len, is_phrase_start}:
[{a, Det, TRUE, 32, 1, TRUE}, {brief, N, TRUE, 34, 5, FALSE},
{press, N, TRUE, 40, 5, FALSE}, {conference, N, TRUE, 46, 10, FALSE}]
```

Related Topics

AUTO_LEXER Language Support

At index time, $\texttt{AUTO_LEXER}$ automatically detects the language of the document, and tokenizes and stems the document appropriately.

CREATE POLICY

Creates a policy to use with the $CTX_DOC.POLICY_*$ procedures, certain Oracle Data Mining procedures, and the in-memory Text index.

POLICY_PART_OF_SPEECH

9.14 POLICY PART OF SPEECH

Extracts part of speech information for each word in a body of text.

POLICY NOUN PHRASES has the list of supported languages.

Syntax

policy_name

Specify the policy name created with CTX_DDL.CREATE_POLICY. If the specified policy includes a sectioning preference, the API will honor the sectioning preference. For instance, if HTML sectioning is specified, HTML tags will be removed before processing the input document.

document

A body of text for which the languages are to be extracted. The text is assumed to be plain text with UTF-8 character encoding.

restab

Specify the name of the CLOB locator. The query returns a table with the result of the noun phrase extraction. For each word, the following attributes are also returned:

- pos_tags: the part of speech tags for this word. There can be multiple part of speech tags
 with the most likely tag listed first.
- offset: offset of the word in the input string
- length: length of the word in the input string.
- is in lexicon: Indicates whether the word is in the lexicon.

language

Specify the language. See the list of supported languages in this section. If this parameter is null, the language will be automatically detected. There is a cost associated with language detection.

format

The format of the input text.

charset

The character set of the input text.

Example for POLICY_PART_OF_SPEECH

The example in this section uses the abbreviations shown in Table 9-1.

```
set serveroutput on;
declare
 the nps ctx doc.part of speech tab;
begin
   ctx doc.policy part of speech (policy name => 'rv policy 21',
                                document => 'The mayor of Chicago is giving
                                             a brief press conference',
                                 restab => the nps,
                                 disambiguate tags => false,
                                 language => 'english');
for i in 1..the nps.count loop
 dbms output.put('word:' || the nps(i).word || ',pos:[');
  for j in 1..the nps(i).pos_tags.count loop
   dbms output.put(the nps(i).pos tags(j) || ',');
 end loop;
 dbms output.put line(']');
end loop;
end;
```



Output for this example:

```
word:The,pos:[Det,]
word:mayor,pos:[N,]
word:of,pos:[Prep,]
word:Chicago,pos:[propN,]
word:is,pos:[V,]
word:giving,pos:[N,V,Adj,]
word:a,pos:[Det,]
word:brief,pos:[N,V,Adj,]
word:press,pos:[N,V,]
word:conference,pos:[N,V,]
```

Related Topics

"POLICY NOUN PHRASES"

"Custom Dictionary Valid Parts-of-Speech (case sensitive)"

9.15 POLICY SNIPPET

Displays marked-up keywords in context. The returned text contains either the words that satisfy a word query or the themes that satisfy an ABOUT query. This version of the CTX_DOC.SNIPPET procedure does not require an index.

Syntax

Syntax 1

Syntax 2



policy_name

Specify the name of a policy created with CTX_DDL.CREATE_POLICY.

document

Specify the document in which to search for keywords.

text_query

Specify the original query expression used to retrieve the document. If NULL, no highlights are generated.

If text_query includes wildcards, stemming, fuzzy matching which result in stopwords being returned, POLICY SNIPPET does not highlight the stopwords.

If text guery contains the threshold operator, the operator is ignored.

language

Specify the language of the document. Use an Oracle Text supported language value as you would in the language column of the base table. See MULTI_LEXER in Oracle Text Indexing Elements.

format

Specify the format of the document. Use an Oracle Text supported format value, either TEXT, BINARY or IGNORE as you would specify in the format column of the base table. For more information, see the format column description in "CREATE INDEX".

charset

Specify the character set of the document. Use an Oracle Text supported value as you would specify in the charset column of the base table. See "Filter Types".

starttag

Specify the start tag for marking up the guery keywords. Default is ''.

endtag

Specify the end tag for marking up the query keywords. Default is ''.

entity_translation

Specify if you want HTML entities to be translated. The default is TRUE, which means the special entities (<, >, and &) are translated into their alternate forms ('<', '>', and '&') when output by the procedure. However, special characters in the markup tags generated by DOC.POLICY SNIPPET will not be translated.

separator

Specify the string separating different returned fragments. Default is '<*b*>'...</*b*>'.

radius

Specify the number of characters to be shown on either side of the hit query in a segment. The character count before the hit query begins on the first character of the first hit query displayed in a segment. Accordingly, the character count after the hit query begins on the last character of the last hit query displayed on a specific segment. Two segments are merged into one if their radii overlap. The displayed number of characters on each side may be modified by +/-10 chars to best match the beginning or ending of a sentence or word.

Special attention is required for the value 0. When specified, the radius is set to automatic and varies between sentences. A best guess of the results is displayed, which attempts to match a full sentence. Note that the length of the radius on each side of the hit query will most likely significantly differ.



The default value is 25.

max_length

Specify the maximum length of the snippet output in characters. This value is currently upperbounded by the current return type of CTX_DOC.SNIPPET and

CTX_DOC.POLICY_SNIPPET (VARCHAR2). Should the output be longer than the return type VARCHAR2, the result will be truncated.

The default value for max length is 250.



If you set max_length value to a very low value, no snippet may be generated. For example, if max_length is set to 0 or if max_length is lower than the length of query tokens themselves, no snippet may be generated at all.

Limitations

CTX DOC.POLICY SNIPPET does not support the use of query templates.

CTX_DOC.POLICY_SNIPPET displays marked-up keywords in context when used with NULL_SECTION_GROUP. However, there are limitations when using this procedure with XML documents. When used with XML_SECTION_GROUP or AUTO_SECTION_GROUP, the XML structure is ignored and user-specified tags are stripped out, which results in parts of surrounding text to be included in the returned snippet.

Related Topics

"SNIPPET"

"MARKUP"

9.16 POLICY_STEMS

Extracts stems for each word in a body of text. This procedure is for use with AUTO_LEXER. This procedure can only use the languages supported by AUTO_LEXER, which are listed under "POLICY LANGUAGES".

Syntax

```
exec CTX_DOC.POLICY_STEMS (
   policy_name    in varchar2,
   document    in varchar2 | CLOB,
   restab     in out nocopy ctx_doc.stem_group_tab,
   language    in varchar2 default NULL,
   format     in varchar2 default NULL,
   charset    in varchar2 default NULL
```

policy name

A policy that was previously created using the CTX_DDL.CREATE_POLICY method. If the specified policy includes a HTML_SECTION_GROUP sectioning preference, the API will honor the sectioning preference. For instance, if HTML sectioning is specified, HTML tags will be removed before processing the input document.

Note that the policy must use AUTO LEXER only.

document

A body of text for which the languages are to be extracted. The text is assumed to be plain text with UTF-8 character encoding.

restab

The result of the stem extraction process. The returned values in the PL/SQL table will have one cell for each word in the input string document. Each word can be a multi-word as determined by the lexer. For each word, all the stems (including all alternate stems) are returned. For each stem, the offset and the length (in the input string) of the word for which this is a stem is returned. Additionally, for each stem, a Boolean value is returned that indicates if the stem was found in the lexicon.

```
stem group tab is a table of stem group records.
```

language

The language of the input text. The language string can be one of the values specified in the previous section on language extraction. If this parameter is null, the language will be automatically detected. There is a cost associated with language detection. So, if the language is known, it is best to supply the language value. See "POLICY_LANGUAGES" for the list of languages.

format

The format of the input text.

charset

The character set of the input text.

Restrictions and Notes

The stem extraction process supports certain nonstandard word forms—e.g. capitalization errors—as well as standard forms, and thus can be used to process informal or imperfect text (such as email, online documents, or queries). It also handles some variations in the text including case variation, hyphenation and unaccented characters among others.

The stem extraction process does not break compound words, but instead separates compound words with a # character. Such compound words are common in German. For instance, the German compound word *Bildungsroman* (from Bildung "education" and Roman "novel") yields a single stem Bildungs#roman instead of two stems *Bildungs* and *roman*.

Related Topics

```
"POLICY_LANGUAGES"
"AUTO_LEXER"
"CREATE POLICY"
```

9.17 POLICY_THEMES

Generates a list of themes for a document. With this procedure, no CONTEXT index is required.

Syntax



```
charset in VARCHAR2 default NULL
```

);

policy_name

Specify the policy you create with CTX_DDL.CREATE_POLICY.

document

Specify the document for which to generate a list of themes.

restab

Specify the name of the theme tab PL/SQL index-by-table type.



"THEMES" for more information about the structure of the theme_tab type.

full_themes

Specify whether this procedure generates a single theme or a hierarchical list of parent themes (full themes) for each document theme.

Specify TRUE for this procedure to write full themes to the THEME column of the result table. Specify FALSE for this procedure to write single theme information to the THEME column of the result table. This is the default.

num themes

Specify the maximum number of themes to retrieve. For example, if you specify 10, up to first 10 themes are returned for the document. The default is 50.

If you specify 0 or NULL, this procedure returns all themes in a document. If the document contains more than 50 themes, only the first 50 themes show conceptual hierarchy.

language

Specify the language of the document. Use an Oracle Text supported language value as you would in the language column of the base table. See "MULTI_LEXER" in Oracle Text Indexing Elements.

format

Specify the format of the document. Use an Oracle Text supported format value, either TEXT, BINARY or IGNORE as you would specify in the format column of the base table. For more information, see the format column description in "CREATE INDEX" in Oracle Text SQL Statements and Operators .

charset

Specify the character set of the document. Use an Oracle Text supported value as you would specify in the charset column of the base table. See "Filter Types".

Example

Create a policy:

```
exec ctx ddl.create policy('mypolicy');
```

Run themes:

```
declare
  la     varchar2(200);
  rtab     ctx_doc.theme_tab;
begin
    ctx_doc.policy_themes('mypolicy',
```



```
'To define true madness, What is''t but to be nothing but mad?', rtab);
for i in 1..rtab.count loop
   dbms_output.put_line(rtab(i).theme||':'||rtab(i).weight);
end loop;
end;
```

9.18 POLICY_TOKENS

Generate all index tokens for document. With this procedure, no CONTEXT index is required.

Syntax

policy_name

Specify the policy name created with CTX DDL.CREATE POLICY.

document

Specify the document for which to generate tokens.

restab

Specify the name of the token tab PL/SQL index-by-table type.

The tokens returned are those tokens which are inserted into the index for the document. Stop words are not returned. Section tags are not returned because they are not text tokens.



"TOKENS" of this chapter for more information about the structure of the $token_tab$ type

language

Specify the language of the document. Use an Oracle Text supported language value as you would in the language column of the base table. See "MULTI_LEXER" in Oracle Text Indexing Elements.

format

Specify the format of the document. Use an Oracle Text supported format value, either TEXT, BINARY or IGNORE as you would specify in the format column of the base table. For more information, see the format column description in "CREATE INDEX".

charset

Specify the character set of the document. Use an Oracle Text supported value as you would specify in the charset column of the base table. See "Filter Types".

thes_name

Specify the thesaurus name. If you do not specify a name, no synonyms or broader terms for index tokens will be generated.

To use the system default thesaurus, specify DEFAULT.

thes toktype

Specify SYN to generate synonyms. Alternatively, specify BT to generate broader terms of index tokens. By default, only synonyms are generated. To use this parameter, you must first specify the thesaurus name using the thesaurus name parameter.

Example 1

Get tokens:

Example 2

This example uses thesaurus support to generate synonyms for tokens:

```
declare
  rtab   ctx_doc.token_tab;
begin
   ctx_doc.policy_tokens('mypolicy','the lazy dog',rtab,thes_name =>'animals');
  for i in 1..rtab.count loop
    dbms_output.put_line(rtab(i).token||'a'||rtab(i).thes_tokens);
  end loop;
end;
```

9.19 SENTIMENT

Use this procedure to perform sentiment analysis for a document, determine a sentiment score for each topic within the document, and populate the results into a result table.

The mandatory inputs to this procedure include the name of a text index associated with the document set and the text key, which is a unique identifier that identifies each document. After sentiment classification is performed, the text segments from the document and their associated sentiment scores are populated into the result table. The sentiment score is a value between -100 and 100.

The result table must exist before you run this procedure. An error is returned if the result table does not exist or if the specified topic is null.

If the specified topic is not present in the document, then a default snippet and sentiment score of zero are written into the result table. If no sentiment classifier is specified, then the default sentiment classifier is used. The default classifier is only available when using AUTO_LEXER.

Syntax

```
SENTIMENT(
   index_name IN VARCHAR2,
   textkey IN VARCHAR2,
   topic IN VARCHAR2,
```



```
restab IN VARCHAR2,
clsfier_name IN VARCHAR2 default NULL,
ttype IN VARCHAR2 default 'EXACT',
radius IN NUMBER default 50,
max_inst IN NUMBER default 5,
starttag IN VARCHAR2 default '',
endtag IN VARCHAR2 default '',
use_saved_copy IN NUMBER default 0);
```

Most parameters in Sentiment are also used in Sentiment_AGGREGATE. For a description of parameters common to Sentiment and Sentiment_AGGREGATE, refer to SENTIMENT_AGGREGATE.

restab

Specify the name of the result table that will be populated with generated results. The table must exist and you must have INSERT permissions on the table. The table must have two columns, snippet of data type CLOB and score of data type NUMBER.

starttag

Specify the character(s) to be inserted to indicate the start of a highlighted term.

endtag

Specify the character(s) to be inserted to indicate the end of a highlighted term.



Oracle Text Application Developer's Guide for an example of using the ${\tt SENTIMENT}$ procedure

9.20 SENTIMENT AGGREGATE

Use this procedure to perform sentiment analysis and return a single aggregate sentiment score per document. The aggregate sentiment score is a value between -100 and 100.

You specify search keywords as part of a text query and then identify a sentiment associated with the topics in the document.

The mandatory inputs for this procedure include the name of a text index associated with the document set and the text key, which is a unique identifier that identifies each document. If no sentiment classifier is specified, then the default sentiment classifier is used. The default classifier is only available when using AUTO_LEXER.

If the specified topic keyword is not found within the document, then a sentiment score of zero is returned. If no topic is specified, then the aggregate sentiment score for the entire document is returned.





Avoid using AUTO_LEXER with user-defined classifiers as this may provide inconsistent sentiment scores.

Syntax

```
SENTIMENT_AGGREGATE(
    index_name IN VARCHAR2,
    textkey IN VARCHAR2,
    topic IN VARCHAR2 default NULL,
    clsfier_name IN VARCHAR2 default NULL,
    ttype IN VARCHAR2 default 'EXACT',
    radius IN NUMBER default 50,
    max_inst IN NUMBER default 5,
    use_saved_copy IN NUMBER default 0
) return NUMBER;
```

index name

Specify the name of the CONTEXT index for the text column. This parameter is mandatory.

textkey

Specify the unique identifier (usually the primary key) for the document. The textkey is mandatory and is a single column primary key value.

clsfier name

Specify the name of the sentiment classifier used to perform sentiment analysis. The maximum length supported for a classifier name is 24 bytes. If you do not specify a classifier name, then the default classifier is used.

topic

Specify the topic for which a sentiment score must be generated for this document. If the topic is not specified, then the sentiment score for the entire document is generated.

ttype

Specify the type of search to be performed for this document:

- EXACT: Indicates that the specified search keyword must be searched in the document.
 This is the default setting.
- ABOUT: Indicates that the thesaurus must be used to find words that are related to the search keywords.

radius

Specifies the radius of the surrounding text to be analyzed during sentiment classification. The default value is 50.

The exact amount of text used for analysis varies from case to case because Oracle Text attempts to find the best match text segment with respect to nearby topic keywords, word boundaries, and sentence boundaries.

max inst

Specify the maximum number of instances/occurrences of the topic that must be analyzed. The default value for this parameter is 5.

use_saved_copy

Specify whether to refer to the $\protect\ D$ table to fetch the copy of the document and what action to take when the copy of the document is not available in the $\protect\ D$ table. The default value of this parameter is zero.



Oracle Text Application Developer's Guide for an example of using the SENTIMENT AGGREGATE procedure

9.21 SET KEY TYPE

Use this procedure to set the CTX_DOC procedures to accept either the ROWID or the PRIMARY KEY document identifiers. This setting affects the invoking session only.

Syntax

```
ctx_doc.set_key_type(key_type in varchar2);
```

key_type

Specify either ROWID or PRIMARY_KEY as the input key type (document identifier) for CTX_DOC procedures.

This parameter defaults to the value of the CTX DOC KEY TYPE system parameter.

Note:

- When your base table has no primary key, setting key_type to PRIMARY_KEY is
 ignored. The textkey parameter that you specify for any CTX_DOC procedure is
 interpreted as a ROWID.
- CTX_DOC.SET_KEY_TYPE fails to set PRIMARY_KEY as the input key type for CTX_DOC procedures, if it's PRIMARY_KEY is added to the table post index creation. ORA-20000 error is displayed. The workaround is to drop the index and recreate the index.

Example

The following example sets CTX DOC procedures to accept primary key document identifiers.

```
begin
ctx_doc.set_key_type('PRIMARY_KEY');
ond
```

9.22 SNIPPET

Use the CTX_DOC.SNIPPET procedure to produce a concordance for a document. The output of a snippet is a collection of segments. A concordance is a text fragment that contains a query term with some of its surrounding text. This is also sometimes known as Key Word in Context or KWIC, because it returns query keywords marked up in their surrounding text, which

enables the user to evaluate them in context. The returned text can also contain themes that satisfy an ABOUT query.

For example, a search on *brillig* and *slithey* might return one relevant fragment of a document as follows:

```
'Twas <b>brillig</b>, and the <b>slithey</b> toves did gyre and
```

CTX_DOC.SNIPPET returns one or more most relevant fragments for a document that contains the query term. Because CTX_DOC.SNIPPET returns surrounding text, you can immediately evaluate how useful the returned term is. CTX_DOC.SNIPPET returns the entire document if no words in the returned text are marked up.

Note that for queries that have predicates used mainly for filtering documents at query time, the predicates are ignored during SNIPPET generation. The following predicates are treated as filter predicates for this purpose: SDATA, HASPATH, and WITHIN/INPATH searching inside xml attributes.

```
See Also:
```

CTX_DOC.POLICY_SNIPPET for a policy-based version of this procedure

Syntax

Syntax 1

Syntax 2



index name

Specify the name of the index for the text column.

textkey

Specify the unique identifier (usually the primary key) for the document.

The textkey parameter can be as follows:

- A single column primary key value
- An encoded specification for a composite (multiple column) primary key. When textkey is a
 composite key, you must encode the composite textkey string using the CTX_DOC.PKENCODE
 procedure.
- The rowid of the row containing the document

Use CTX DOC. SET KEY TYPE to toggle between primary key and rowid identification.

text_query

Specify the original query expression used to retrieve the document. If NULL, no highlights are generated.

If text_query includes wildcards, stemming, fuzzy matching which result in stopwords being returned, SNIPPET does not highlight the stopwords.

If text guery contains the threshold operator, the operator is ignored.

starttag

Specify the start tag for marking up the query keywords. Default is ''.

endtag

Specify the end tag for marking up the query keywords. Default is ''.

entity translation

Specify if you want HTML entities to be translated. The default is TRUE, which means that the special entities (<, >, and &) are translated into their alternative forms ('<', '>', and '&') when output by the procedure. However, special characters in the markup tags that are generated by CTX DOC.SNIPPET will not be translated.

separator

Specify the string separating different returned fragments. Default is '<*b*>...</*b*>'.

radius

Specify the number of characters to be shown on either side of the hit query in a segment. The character count before the hit query begins on the first character of the first hit query displayed in a segment. Accordingly, the character count after the hit query begins on the last character of the last hit query displayed on a specific segment. Two segments are merged into one if their radii overlap. The displayed number of characters on each side may be modified by +/-10 chars to best match the beginning or ending of a sentence or word.

Special attention is required for the value 0. When specified, the radius is set to automatic and varies between sentences. A best guess of the results is displayed, which attempts to match a full sentence. Note that the length of the radius on each side of the hit query will most likely significantly differ.

The default value is 25.

max length

Specify the maximum length of the snippet output in characters. This value is currently upper-bounded by the current return type of CTX_DOC.SNIPPET and

CTX_DOC.POLICY_SNIPPET (VARCHAR2). Should the output be longer than the return type VARCHAR2, the result will be truncated. The default value for max length is 250.



If you set max_length value to a very low value, no snippet may be generated. For example, if max_length is set to 0 or if max_length is lower than the length of query tokens themselves, no snippet may be generated at all.

use_saved_copy

Specify whether to refer to the \$D table to fetch the copy of the document, and what action to take when the copy of the document is not available in the \$D table. The default value is CTX DOC.SAVE COPY FALLBACK.

You can specify one of the following values for the use saved copy parameter:

- CTX_DOC.SAVE_COPY_FALLBACK: Fetch the copy of the document from the \$D table. If the copy of the document is not present in the \$D table, then fetch the document from the data store.
- CTX_DOC.SAVE_COPY_ERROR: Fetch the copy of the document from the \$D table. If the copy
 of the document is not present in the \$D table, then show an error message. Specify this
 value when you want to implement a specific fallback logic when the copy of the
 document is not available in the \$D table.
- CTX DOC.SAVE COPY IGNORE: Always fetch the document from the data store.

Example

```
create table tdrbhk01 (id number primary key, text varchar2(4000));
insert into tdrbhk01 values (1, 'Oracle Text adds powerful search
and intelligent text management to the Oracle
database. Complete. You can search and manage documents, web pages,
catalog entries in more than 150 formats in any language. Provides a
complete text query language and complete character support. Simple. You
can index and search text using SQL. Oracle Text Management can be done
using Oracle Enterprise Manager - a GUI tool. Fast. You can search
millions of documents, document, web pages, catalog entries using the
power and scalability of the database. Intelligent. Oracle Text''s
unique knowledge-base enables you to search, classify, manage
documents, clusters and summarize text based on its meaning as well as
its content. ');
create index tdrbhk01x on tdrbhk01(text) indextype is ctxsys.context;
create or replace function my snippet wrapper(
 key in varchar2,
 query in varchar2,
 radius in number,
 max length in number) return varchar2 is
 buff varchar2(4000);
  buff := ctx doc.snippet('tdrbhk01x', key, query, '<b>', '<b>', true, '<b>'.<b>',
radius, max length);
  return buff;
 end;
show errors;
select my snippet wrapper('1','Oracle', 10, 100) from dual;
The result looks something like this:
CTX DOC.SNIPPET('TDRBHK01X','1','SEARCH|CLASSIFY')
```



```
Text's unique knowledge-base enables you to <b>search</b>, <b>classify</b>, manage documents, clusters and summarize
```

Limitations

CTX DOC. SNIPPET does not support the use of query templates.

CTX_DOC.SNIPPET displays marked-up keywords in context when used with NULL_SECTION_GROUP. However, there are limitations when using this procedure with XML documents. When used with XML_SECTION_GROUP or AUTO_SECTION_GROUP, the XML structure is ignored and user-specified tags are stripped out, which results in parts of surrounding text to be included in the returned snippet.

Related Topics

```
"POLICY_SNIPPET"
"HIGHLIGHT "
"MARKUP "
```

9.23 THEMES

Use the CTX_DOC.THEMES procedure to generate a list of themes for a document. You can store each theme as a row in either a result table or an in-memory PL/SQL table that you specify.

Syntax 1: In-Memory Table Storage

Syntax 2: Result Table Storage

```
IN VARCHAR2,
textkey IN VARCHAR2,
restab IN VARCHAR2,
query_id IN NUMBER DEFAULT 0,
full_themes IN BOOLEAN DEFAULT FALSE,
num_themes IN NUMBER DEFAULT 50,
use_saved_copy IN NUMBER DEFAULT CTX_DOC.SAVE_COPY_FALLBACK);
```

index_name

Specify the name of the index for the text column.

textkey

Specify the unique identifier (usually the primary key) for the document. The textkey parameter can be as follows:

A single column primary key value



- An encoded specification for a composite (multiple column) primary key. When textkey is a composite key, you must encode the composite textkey string using the CTX DOC.PKENCODE procedure.
- The rowid of the row containing the document

Toggle between primary key and rowid identification using CTX DOC.SET KEY TYPE.

restab

You can specify this procedure to store results to either a table or to an in-memory PL/SQL table.

To store results in a table, specify the name of the table.



"Theme Table" in Oracle Text Result Tables

To store results in an in-memory table, specify the name of the in-memory table of type THEME TAB. The THEME TAB datatype is defined as follows:

```
type theme_rec is record (
    theme varchar2(2000),
    weight number
);

type theme_tab is table of theme_rec index by binary_integer;

CTX DOC.THEMES clears the THEME TAB you specify before the operation.
```

query id

Specify the identifier used to identify the row(s) inserted into restab.

full themes

Specify whether this procedure generates a single theme or a hierarchical list of parent themes (full themes) for each document theme.

Specify TRUE for this procedure to write full themes to the THEME column of the result table. Specify FALSE for this procedure to write single theme information to the THEME column of the result table. This is the default.

num themes

Specify the maximum number of themes to retrieve. For example, if you specify 10, then up to the first 10 themes are returned for the document. The default is 50.

If you specify 0 or NULL, then this procedure returns all themes in a document. If the document contains more than 50 themes, then only the first 50 themes show conceptual hierarchy.

use saved copy

Specify whether to refer to the \$D table to fetch the copy of the document, and what action to take when the copy of the document is not available in the \$D table.

You can specify one of the following values for the use saved copy parameter:

- CTX_DOC.SAVE_COPY_FALLBACK: Fetch the copy of the document from the \$D table. If the copy of the document is not present in the \$D table, then fetch the document from the data store.
- CTX_DOC.SAVE_COPY_ERROR: Fetch the copy of the document from the \$D table. If the copy of the document is not present in the \$D table, then show an error message. Specify this

value when you want to implement a specific fallback logic when the copy of the document is not available in the \$D table.

CTX_DOC.SAVE_COPY_IGNORE: Always fetch the document from the data store.

The default value is CTX_DOC.SAVE_COPY_FALLBACK.

Examples

In-Memory Themes

The following example generates the first 10 themes for document 1 and stores them in an inmemory table called the_themes. The example then loops through the table to display the document themes.

```
declare
  the_themes ctx_doc.theme_tab;

begin
  ctx_doc.themes('myindex','1',the_themes, num_themes=>10);
  for i in 1..the_themes.count loop
   dbms_output.put_line(the_themes(i).theme||':'||the_themes(i).weight);
  end loop;
end;
```

Theme Table

The following example creates a theme table called CTX THEMES:

Single Themes

To obtain a list of up to the first 20 themes, where each element in the list is a single theme, enter a statement like the following example:

```
begin

ctx_doc.themes('newsindex','34','CTX_THEMES',1,full_themes => FALSE,
    num_themes=> 20);
end;
```

Full Themes

To obtain a list of the top 20 themes, where each element in the list is a hierarchical list of parent themes, enter a statement like the following example:

```
begin
ctx_doc.themes('newsindex','34','CTX_THEMES',1,full_themes => TRUE, num_themes=>20);
end;
```



9.24 TOKENS

Use this procedure to identify all text tokens in a document. The tokens returned are those tokens that are inserted into the index.

Thesaurus support also enables you to generate synonyms or broader terms of the queried index tokens. This feature is useful for implementing document classification, routing, or clustering.

Stopwords are not returned. Section tags are not returned because they are not text tokens.

Syntax 1: In-Memory Table Storage

```
CTX_DOC.TOKENS(index_name IN VARCHAR2,
textkey IN VARCHAR2,
restab IN OUT NOCOPY TOKEN_TAB,
thes_name IN VARCHAR2 DEFAULT NULL,
thes_toktype IN VARCHAR2 DEFAULT 'SYN',
use_saved_copy IN NUMBER DEFAULT CTX_DOC.SAVE_COPY_FALLBACK);
```

Syntax 2: Result Table Storage

```
CTX_DOC.TOKENS(index_name IN VARCHAR2,
textkey IN VARCHAR2,
restab IN VARCHAR2,
thes_name IN VARCHAR2 DEFAULT NULL,
thes_toktype IN VARCHAR2 DEFAULT 'SYN',
query_id IN NUMBER DEFAULT 0,
use_saved_copy IN NUMBER DEFAULT CTX DOC.SAVE COPY FALLBACK);
```

index name

Specify the name of the index for the text column.

textkey

Specify the unique identifier (usually the primary key) for the document.

The textkey parameter can be as follows:

- A single column primary key value
- Encoded specification for a composite (multiple column) primary key. To encode a composite textkey, use the CTX DOC.PKENCODE procedure.
- The rowid of the row containing the document

Toggle between primary key and rowid identification using CTX DOC.SET KEY TYPE.

restab

You can specify that this procedure store results to either a table or to an in-memory PL/SQL table.

The tokens returned are those tokens that are inserted into the index for the document (or row) named with textkey. Stop words are not returned. Section tags are not returned because they are not text tokens.

thes name

Specify the thesaurus name. If you do not specify a thesaurus name, then no synonyms or broader terms will be generated. To use the system default thesaurus, specify DEFAULT.



If you specify thes_name, then the token table must include the THES_TOKENS column, otherwise the CTX_DOC.TOKENS procedure fails with an "ORA-00904: THES_TOKENS: Invalid identifier when thes name parameter is used" error.

thes_toktype

Specify SYN to generate synonyms of index tokens. Alternatively, specify BT to generate broader terms of index tokens. By default, synonyms are generated. To use this parameter, you must first specify a thesaurus name using the thes name parameter.

Specifying a Token Table

To store results to a table, specify the name of the table. Token tables can be named anything, but must include the columns shown in the following table, with names and datatypes as specified.

Table 9-2 Required Columns for Token Tables

Column Name	Туре	Description
QUERY_ID	NUMBER	The identifier for the results generated by a particular call to CTX_DOC.TOKENS (only populated when table is used to store results from multiple TOKEN calls)
TOKEN	VARCHAR2 (255)	The token string in the text.
THES_TOKENS	VARCHAR2 (4000)	Synonyms or broader terms generated using a thesaurus for the token in the ${\tt TOKEN}$ column. These values must be colonseparated.



The THES_TOKENS column is required only if you specify the thes_name argument with the CTX_DOC.TOKENS API.

OFFSET	NUMBER	The position of the token in the document, relative to the start of document which has a position of 1.
LENGTH	NUMBER	The character length of the token.

Specifying an In-Memory Table

To store results to an in-memory table, specify the name of the in-memory table of type TOKEN TAB. The TOKEN TAB datatype is defined as follows:

```
type token_rec is record (
token varchar2(255),
offset number,
length number
);
type token_tab is table of token_rec index by binary_integer;
CTX_DOC.TOKENS clears the TOKEN_TAB you specify before the operation.
```



query id

Specify the identifier used to identify the row(s) inserted into restab.

use saved copy

Specify whether to refer to the \$D table to fetch the copy of the document, and what action to take when the copy of the document is not available in the \$D table.

You can specify one of the following values for the use saved copy parameter:

- CTX_DOC.SAVE_COPY_FALLBACK: Fetch the copy of the document from the \$D table. If the copy of the document is not present in the \$D table, then fetch the document from the data store.
- CTX_DOC.SAVE_COPY_ERROR: Fetch the copy of the document from the \$D table. If the copy of the document is not present in the \$D table, then show an error message. Specify this value when you want to implement a specific fallback logic when the copy of the document is not available in the \$D table.
- CTX DOC.SAVE COPY IGNORE: Always fetch the document from the data store.

The default value is CTX DOC. SAVE COPY FALLBACK.

Example

In-Memory Tokens

The following example generates the tokens for document 1 and stores them in an in-memory table, declared as the_tokens. The example then loops through the table to display the document tokens.

```
declare
  the_tokens ctx_doc.token_tab;

begin
  ctx_doc.tokens('myindex','1',the_tokens);
  for i in 1..the_tokens.count loop
   dbms_output.put_line(the_tokens(i).token);
  end loop;
end;
```



CTX_ENTITY Package

The CTX_ENTITY PL/SQL package is used to locate and classify words and phrases into categories, such as persons or companies.

CTX ENTITY contains the following stored procedures and functions.

Name	Description
ADD_EXTRACT_RULE	Adds a single extraction rule to an extraction policy.
ADD_STOP_ENTITY	Marks certain entity mentions or entity types as not to be extracted.
COMPILE	Compiles added extraction rules into an extraction policy.
CREATE_EXTRACT_POLICY	Creates an extraction policy to use.
DROP_EXTRACT_POLICY	Drops an extraction policy.
EXTRACT	Generates an XML document describing the entities found in an input document.
IMPORT_DICTIONARY	Imports an entity extraction user dictionary into Oracle Text tables.
REMOVE_EXTRACT_RULE	Removes a single extraction rule from an extraction policy.
REMOVE_STOP_ENTITY	Removes a stop entity from an extraction policy.



The APIs in the CTX_ENTITY package do not support identifiers that are prefixed with the schema or the owner name.

10.1 ADD_EXTRACT_RULE

The ${\tt ADD_EXTRACT_RULE}$ procedure adds a single extraction rule to extract policy. Invokers add rules into their own extraction policy.

Extraction rules have sentence-wide scopes. Extraction rules have to be case-sensitive except for entity types and rule operators in the rule expression. Order of rule addition is not important. Addition of a rule will not be effective until CTX_ENTITY.COMPILE is executed. This procedure issues a commit.

Syntax

policy_name

Specify the policy name.

rule id

Specify a unique rule ID within an extraction policy. The rule ID must be greater than 0.

extraction rule

The rule text in XML format specifies the language, expression, and entities to be extracted. The rule text follows the XML schema as follows:

Where:

- The language attribute of the rule tag specifies the applied language for the rule. The rule will only be applied to documents that are of the specified languages. The language attribute can be left out, or set to "ALL" if the rule is to match on all documents.
- The expression tag contains the posix regular expression that will be used in the matching.
- The comments tag allows users to associate any comments with this user rule.
- The type tag assigns the extracted entity text to a given entity type. The entity type can be one of the Oracle supplied rule types, listed in Table 10-1, or it can be a user-defined type.

Note:

Starting with Oracle Database Release 21c, the extraction rule's XML format has the following changes:

- The refid attribute of the type tag is not supported.
- User-defined types do not need to be prefixed with the letter "x".
- '\c (<type>) ' must be used for using user-defined type and Oracle supplied types in the rules.

Supplied Entity Type	Туре	Explanation	Examples
building	Oracle supplied dictionary	Name of a building	Gallery House
city	Oracle supplied dictionary	Name of a city	New York
company	Oracle supplied dictionary	Name of a company	Oracle Corporation
country	Oracle supplied dictionary	Name of a country	United States
currency	Oracle supplied rule	Currency	Dollar



Supplied Entity Type	Туре	Explanation	Examples
date	Oracle supplied rule	Date	July 4
day	Oracle supplied dictionary	Day	Monday, Tuesday
email_address	Oracle supplied rule	Email address	person@example.c om
geo_political	Oracle supplied dictionary	A political or strategic organization	United Nations
holiday	Oracle supplied dictionary	Name of a country holiday	Labor Day
location_other	Oracle supplied dictionary	Other types of locations	Atlantic Ocean
month	Oracle supplied rule	Month	June, July
non_profit	Oracle supplied dictionary	Non-profit organization	Red Cross
organization_ot her	Oracle supplied dictionary	Other types of organizations	Supreme Court
percent	Oracle supplied rule	Expressed as number and %	10%
person_jobtitle	Oracle supplied dictionary	Person referred to by title	President, Professor
person_name	Oracle supplied rule	Person referred to by name	John Doe
person_other	Oracle supplied dictionary	Other types of persons	Other types of persons (for example, criminal)
phone_number	Oracle supplied rule	Phone number	(123)-456-7890
postal_address	Oracle supplied rule	Postal address	Redwood Shores, CA
product	Oracle supplied dictionary	Name of a product	Oracle Text
region	Oracle supplied dictionary	Name of a region	North America
ssn	Oracle supplied rule	Social Security Number	123-45-6789
state	Oracle supplied dictionary	A state or province	California
time_duration	Oracle supplied rule	A length of time	10 seconds
tod	Oracle supplied rule	Time of day	8:00 AM
url	Oracle supplied rule	Web address	www.example.com
zip_code	Oracle supplied rule	Zip Code	CA 94065



Example 10-1 Defining an extraction rule to find email addresses in documents

The following example shows how to define an extraction rule and associate it with an entity extraction policy. The following rule defines a simple extraction rule for finding email addresses in documents.

```
begin
  ctx_entity.add_extract_rule('pol1', 1,
  '<rule>
     <expression>email is (\w+@\w+\.\w+)</expression>
     <type>email_address</type>
     </rule>');
end;
/
```

Where:

- Given the sentence: "My email address is jdoe@example.com", this extraction rule will extract "jdoe@example.com" as an entity of type email address.
- The rule is added to the extraction policy called pol1.
- The rule is added with rule ID of 1.
- This XML description of the rule is as follows:
 - The language attribute of the rule tag is left empty, so the rule will apply to all languages.
 - The expression tag contains the regular expression to use in the extraction.

Example 10-2 Defining an extraction rule to find phone numbers in documents

The following rule defines a simple extraction rule for finding phone numbers in documents:

```
begin
   ctx_entity.add_extract_rule('pol1', 2,
   '<rule language="english">
        <expression>(\(d{3}\\)\\d{3}-\d{3}-\d{4})</expression>
        <comments>Rule for phone numbers</comments>
        <type>email_address</type>
        </rule>');
end;
/
```

Where:

- Given the sentence: "I can be contacted at (123) 456-7890", this extraction rule will extract "(123) 456-7890" as an entity of type phone number.
- The rule is added to the extraction policy called pol1.
- The rule is added with rule ID of 2.
- The XML description of the rule is as follows:
 - The language attribute of the rule tag is set to english, so the rule will only apply to English documents.
 - The expression tag contains the regular expression to use in the extraction.
 - Explanatory comments are associated with this rule.



Example 10-3 Defining an extraction rule using user-defined type

The following example shows how to define an extraction rule using an user-defined type to search for entities in a document:

```
begin
   ctx_entity.add_extract_rule('pol1', 1,
   '<rule>
        <expression>([a-z]+)</expression>
        <type>my_type</type>
        </rule>');
end;

begin
   ctx_entity.add_extract_rule('pol1', 2,
   '<rule>
        <expression>(\d\c(my_type)?\s^\c(my_type))</expression>
        <type>type_comp</type>
        <comments>Rule with nested type</comments>
        </rule>');
end;
/
```

10.2 ADD_STOP_ENTITY

This procedure is used to mark certain entity mentions or entity types as not to be extracted. Invokers add stop entities to their own extraction policy. It does not take effect until after CTX_ENTITY.COMPILE is run. Either entity_name or entity_type can be NULL, but not both. If one stop entity is a subset of another, it will be marked as a subset after CTX_ENTITY.COMPILE, and not used in extraction. This procedure issues a commit.

Syntax

policy_name

Specify the policy name of the stop entity that is to be added.

entity_name

Specify the entity name to be listed as a stop entity. If <code>entity_type</code> is <code>NULL</code>, all mentions with this <code>entity_name</code> will be listed as stop entities. It is case-sensitive.

entity_type

If <code>entity_name</code> is <code>NULL</code>, this will specify an entire entity type to be listed as stop entity. If <code>entity_name</code> is not <code>NULL</code>, this will specify only the mention <code><entity_type</code>, <code>entity_name></code> as a stop entity. It is case-insensitive. The maximum byte length is 4000 bytes.

comments

The maximum byte length is 4000 bytes.

Example

The following example adds a stop entity corresponding to all persons. After compilation, extraction will not report any mentions of entity type person.

```
exec ctx entity.add stop entity('pol1', NULL, 'person');
```

The following example adds a stop entity corresponding to <'person', 'john doe'>. After compilation, extraction will not report any mentions of the pair <'person', 'john doe'>. This stop entity is actually a subset of the first stop entity added. It will be marked subset in the CTX_USER_EXTRACT_STOP_ENTITIES view, and will not be used in extraction.

```
exec ctx_entity.add_stop_entity('pol1', 'john doe', 'person');
```

The following example adds a stop entity corresponding to all mentions of ford. After compilation, extraction will not report any mentions of the entity ford, irrespective of the entity type of the mention. For example, if a rule matches ford to a person, the extraction will not report this match. If a rule matches ford to a company, the extraction will again not report this match.

```
exec ctx_entity.add_stop_entity('pol1', 'ford', NULL);
```

Related Topics

"COMPILE"

"CTX USER EXTRACT STOP ENTITIES"

10.3 COMPILE

This procedure compiles added extraction rules into an extraction policy. It can also be used to compile added stop entities into an extraction policy. Users have to invoke this procedure if they have added any rules or stop entities to their policy.

Invokers compile rules and stop entities into their own extraction policy. Users can choose to compile added rules, added stop entities, or both.

After compilation, the CTX_USER_EXTRACT_RULES, CTX_USER_EXTRACT_STOP_ENTITIES, and CTX_USER_EXTRACT_TYPE views will show which rules, stop entities, and types are being used in the entity extraction.

Syntax

policy_name

Specify the policy name that is to be compiled.

compile choice

Specify the entity name to be listed as a stop entity. If <code>entity_type</code> is <code>NULL</code>, all mentions with this <code>entity_name</code> will be listed as stop entities. It is case-sensitive.



The options are <code>COMPILE_ALL</code>, <code>COMPILE_RULES</code>, and <code>COMPILE_STOP_ENTITIES</code>. <code>COMPILE_ALL</code> compiles both rules and stop entities. <code>COMPILE_RULES</code> compiles only rules. <code>COMPILE_STOP_ENTITIES</code> compiles only stop entities.

locking

The maximum byte length is 4000 bytes. Configure how COMPILE deals with the situation where another COMPILE is already running on the same policy.

The options for locking are:

CTX ENTITY.LOCK WAIT

If another compile is running, wait until the running compile is complete, then begin compile. (In the event of not being able to get a lock, it will wait forever and ignore the maxtime setting.).

• CTX ENTITY.LOCK NOWAIT

If another compile is running, immediately returns without error.

CTX ENTITY.LOCK NOWAIT ERROR

If another sync is running, error "DRG-51313: timeout while waiting for DML or optimize lock" is raised.

storing

The default value is TRUE. The data used in entity extraction is stored to improve the entity extraction's performance. Specify FALSE to stop storing the data used in entity extraction.

Example

The following example compiles the policy using the default setting:

```
exec ctx entity.compile('pol1');
```

The following example compiles only the stop entities for the policy:

```
exec ctx_entity.compile('pol1', CTX_ENTITY.COMPILE STOP ENTITIES);
```

The following example compiles both rules and stop entities. If a lock exists, the function returns immediately, but does not raise an error.

Related Topics

```
"CTX_USER_EXTRACT_RULES"
"CTX_USER_EXTRACT_STOP_ENTITIES"
"CTX_USER_EXTRACT_TYPE"
```

10.4 CREATE_EXTRACT_POLICY

The CREATE_EXTRACT_POLICY procedure creates an extraction policy to use. This policy can only be used by the policy owner.

Syntax

policy_name

Specify the name of the new extraction policy.

lexer

Specify the name of the lexer preference. Only AUTO_LEXER is supported. If not specified, CTXSYS.DEFAULT_EXTRACT_LEXER will be used. The attributes index_stems and deriv_stems are not allowed.

include_supplied_rules

Specify whether Oracle-supplied rules are included in entity extraction. If false, automatic acronym resolution will be turned off. The default is true.

include_supplied_dictionary

Specify whether the Oracle-supplied dictionary is included in entity extraction. The default is true.

Examples

The following example creates an extraction policy using the default settings. By default, the Oracle-supplied features, such as rules and dictionary, are enabled.

```
exec CTX_ENTITY.CREATE_EXTRACT_POLICY('pol1');
```

The following example creates an extraction policy that explicitly specifies certain parameters. It specifies the lexer to be used as mylex, which must be an AUTO_LEXER preference. It also includes the Oracle-supplied rules, but disables the Oracle-supplied dictionary.

```
exec CTX ENTITY.CREATE EXTRACT POLICY('pol2', 'mylex', TRUE, FALSE);
```

Related Topics

"AUTO_LEXER"

"CTXSYS.DEFAULT EXTRACT LEXER"

10.5 DROP EXTRACT POLICY

The DROP_EXTRACT_POLICY procedure drops an extraction policy. These policies can only be dropped by the policy owner. This procedure issues a commit.

Syntax

policy name

Specify the name of the extraction policy to be dropped.

Example

The following example drops the extraction policy pol2:

```
exec ctx_entity.drop_extract_policy('pol2');
```



10.6 EXTRACT

The EXTRACT procedure runs entity extraction on a given document and generates an XML document describing the entities found in the document.

The XML document will give the entity text, type, and location of the entity in the document. The extraction will use the settings (rules, stop entities, and dictionary) defined in the given extraction policy.

Entity type names in the result will be uppercased. Invokers can run extraction using their own extraction policy.

Before execution, you have to issue CTX ENTITY. COMPILE.

Syntax

policy_name

Run extraction using the given policy.

document

The input document to run extraction on.

If entity_type is NULL, all mentions with this entity_name will be listed as stop entities. It is case-sensitive.

language

Only English is supported.

result

A CLOB containing the XML description of the entities extracted from the document. If <code>entity_type</code> is <code>NULL</code>, all mentions with this <code>entity_name</code> will be listed as stop entities. It is case-sensitive.

entity_type_list

Specify that extraction will only consider a subset of entity types. The <code>entity_type_list</code> is a comma-delimited list. If the <code>entity_type_list</code> is not specified, the entity extraction will consider all entity types.

Example

The following example shows the results of entity extraction on an example document. Suppose that we have created an extraction policy called pol1, and we are given the input document:

```
Sam A. Schwartz retired as executive vice president of Org Inc. in New York.
```

We then call the <code>ctx_entity.extract</code> procedure to generate an XML document containing the entities in this document. We insert the results <code>CLOB</code> into a table called <code>entities</code> for future viewing.



```
declare
  myresults clob;
begin
  select txt into mydoc from docs where id=1;
  ctx_entity.extract('p1', mydoc, null, myresults);
  insert into entities values(1, myresults);
  commit;
  end;
/
```

Then we can examine the extracted entities from the entities table. Note that each entity is tagged with its location in the input document, as well as the source used to classify the entity.

```
<entities>
<entity id="0" offset="75" length="8" source="SuppliedDictionary">
<text>New York</text>
<type>city</type>
</entity>
<entity id="1" offset="55" length="16" source="SuppliedRule">
<text>Org Inc.</text>
<type>company</type>
</entity>
<entity id="2" offset="27" length="24" source="SuppliedDictionary">
<text>Sam A. Schwartz</text>
<type>person name</type>
</entity>
<entity id="4" offset="75" length="8" source="SuppliedDictionary">
<text>New York</text>
<type>state</type>
</entity>
</entities>
```

10.7 IMPORT_DICTIONARY

Use the CTX_ENTITY.IMPORT_DICTIONARY procedure to import an entity extraction user dictionary into Oracle Text tables.

An import dictionary is an XML containing entries for entities, with their associated types and alternate forms. The XML schema is the same XML schema used by Entity Extraction User Dictionary Loader (ctxload). You can load only one user dictionary per policy.

Syntax

policy_name

Specify the policy name.

data

Specify the XML dictionary.



The XML schema is as follows:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="dictionary">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="entities" type="entityType" maxOccurs="unbounded"/>
    </xsd:sequence>
  <xsd:complexType>
</xsd:element>
<xsd:complexType name="entityType">
  <xsd:sequence>
    <xsd:element name="entity" type="entType" maxOccurs="unbounded"/>
  </xsd:sequence>
  </xsd:attribute name="language" type="xsd:string"/>
</xsd:complexType>
<xsd:complexType name="entType">
  <xsd:sequence>
    <xsd:element name="value" type="xsd:string"/>
    <xsd:element name="type" type="xsd:string" minOccurs="1"</pre>
maxOccurs="unbounded"/>
    <xsd:element name="alternate" type="xsd:string" minOccurs="0"</pre>
maxOccurs="unbounded"/>
   </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

isdrop

Specify whether the current user dictionary must be dropped. The default value is FALSE.

Example 10-4 Importing an Entity Extraction User Dictionary into an Oracle Text Table

This example shows how to import an entity extraction user dictionary into an Oracle Text table using CTX_ENTITY.IMPORT_DICTIONARY procedure.

Create an extraction policy using the default settings. By default, the Oracle-supplied features, such as rules and dictionary, are enabled.

```
exec ctx_entity.create_extract_policy('mypol')
```

Import an entity extraction user dictionary and compile the extraction policy. Then, run entity extraction on the input document. You can also specify if the current user dictionary must be dropped.

```
declare
  datadic clob;
  doc    clob;
  res    clob;

begin
  datadic := '<dictionary>
  <entities language="english">
    <entity>
```



```
<value>NewEntry</value>
      <type>MyType</type>
    </entity>
  </entities>
  </dictionary>';
  ctx entity.import dictionary('mypol', datadic);
  ctx entity.compile('mypol');
  doc := 'NEWENTRY';
  ctx entity.extract('mypol', doc, 'english', res);
  dbms output.put line(res);
  -- Dropping dictionary
  ctx entity.import dictionary('mypol', null, isdrop=>true);
 ctx entity.compile('mypol');
 ctx entity.extract('mypol', doc, 'english', res);
  dbms output.put line(res);
end;
```

Related Topics

"Entity Extraction User Dictionary Loader (ctxload)"

"COMPILE"

"CREATE EXTRACT POLICY"

"EXTRACT"

10.8 REMOVE_EXTRACT_RULE

The REMOVE_EXTRACT_RULE procedure removes an extraction rule from the specified policy given a rule_id. Only the owner of the specified policy can remove an extraction rule from the policy. Removal of the extraction rule will be in effect after running CTX_ENTITY.COMPILE.

Syntax

policy name

Remove the extraction rule from the specified policy.

rule id

Specify the rule ID of the extraction rule to be removed.

Example

The following example removes the extraction rule with ID 1 from the policy pol1:

```
exec ctx_entity.remove_extract_rule('pol1', 1);
```

10.9 REMOVE_STOP_ENTITY

The REMOVE_STOP_ENTITY procedure removes a stop entity from an extraction policy. Only the owner of the specified policy can remove a stop entity from the policy. Removal of the stop entity will be in effect after running CTX_ENTITY.COMPILE. Either the entity_name or entity type can be null, but not both.

Syntax

policy name

Remove the stop entity from the specified policy.

entity name

Specify the name to be removed from the stop entity list. The <code>stop_entity</code> must have already been added to the <code>stop</code> entity list using <code>CTX</code> ENTITY.ADD_STOP_ENTITY.

entity_type

Specify the type of entity to be removed from the stop entity list. The stop_entity must have already been added to the stop entity list using CTX ENTITY.ADD_STOP_ENTITY.

Example

```
exec ctx entity.remove stop entity('pol1', NULL, 'person name');
```

The example statement removes the stop entity corresponding to all mentions of the entity_type person_name from the policy pol1. After execution, this stop entity will be marked as "to be deleted" in the CTX_USER_EXTRACT_STOP_ENTITIES view. The removal of the stop entity will take effect once the user runs CTX_ENTITY.COMPILE.

Related Topics

```
"COMPILE"

"ADD_STOP_ENTITY"

"CTX USER EXTRACT STOP ENTITIES"
```

CTX_OUTPUT Package

This chapter provides reference information for using the CTX OUTPUT PL/SQL package.

 ${\tt CTX_OUTPUT}\ contains\ the\ following\ stored\ procedures:$

Name	Description
ADD_EVENT	Adds an event to the index log.
ADD_TRACE	Enables tracing.
DISABLE_QUERY_STATS	Turns off the gathering of query stats for the index.
ENABLE_QUERY_STATS	Enables gathering of query stats for the index.
END_LOG	Halts logging of index and document services requests.
END_QUERY_LOG	Stops logging queries into a logfile.
GET_TRACE_VALUE	Returns the value of a trace.
LOG_TRACES	Prints traces to logfile.
LOGFILENAME	Returns the name of the current log file.
REMOVE_EVENT	Removes an event from the index log.
REMOVE_TRACE	Disables tracing.
RESET_TRACE	Clears a trace.
START_LOG	Starts logging index and document service requests.
START_QUERY_LOG	Creates a log file of queries.

Note:

The APIs in the CTX_OUTPUT package do not support identifiers that are prefixed with the schema or the owner name.

11.1 ADD EVENT

Use this procedure to add an event to the index log for a more detailed log output or to enable error tracing for Oracle Text errors. Index logs are now appended to the database trace files.

Syntax

CTX OUTPUT.ADD EVENT(event in NUMBER, errnum in NUMBER := null);

event

Specify the type of index event to log. You can add the following events:

• CTX_OUTPUT.EVENT_INDEX_PRINT_ROWID, which logs the rowid of each row as it is indexed. This is useful for debugging a failed index operation.

- CTX_OUTPUT.EVENT_INDEX_PRINT_TOKEN, which prints the each token as it is being indexed.
- CTX_OUTPUT.EVENT_DRG_DUMP_ERRORSTACK, which prints the stack trace for the specified DRG error in the log. An error will be raised if errnum is not specified.



CTX_OUTPUT.EVENT_OPT_PRINT_TOKEN, which prints each token as it is being optimized, and CTX_OUTPUT.EVENT_INDEX_PRINT_TOKEN, which prints each token as it is being indexed, are disabled when using PDB lockdown profile CTX PROTOCOLS.

errnum

Specify the DRG error number for a CTX_OUTPUT.EVENT_DRG_DUMP_ERRRORSTACK event.

Example

```
begin
CTX_OUTPUT.ADD_EVENT(CTX_OUTPUT.EVENT_INDEX_PRINT_ROWID);
end;
```

Related Topics

"REMOVE EVENT"

11.2 ADD_TRACE

Use this procedure to enable a trace. If the trace has not been enabled, this call adds the trace to the list of active traces and resets its value to 0. If the trace has already been enabled, an error is raised.

Syntax

```
CTX OUTPUT.ADD TRACE(trace id BINARY INTEGER);
```

trace id

Specify the ID of the trace to enable. See Table 11-1 for possible trace values.

Notes

Table 11-1 shows the available traces:

Table 11-1 Available Traces

Symbol	ID	Metric
TRACE_IDX_USER_DATASTORE	1	Time spent executing user datastore
TRACE_IDX_AUTO_FILTER	2	Time spent invoking the AUTO_FILTER filter. (Replaces the deprecated TRACE_IDX_INSO_FILTER trace)
TRACE_QRY_XX_TIME	3	Time spent executing the \$X cursor
TRACE_QRY_XF_TIME	4	Time spent fetching from \$X
TRACE_QRY_X_ROWS	5	Total number of rows whose token metadata was fetched from $\$\mathtt{X}$



Table 11-1 (Cont.) Available Traces

Symbol	ID	Metric
TRACE_QRY_IF_TIME	6	Time spent fetching the LOB locator from \$I
TRACE_QRY_IR_TIME	7	Time spent reading \$I LOB information
TRACE_QRY_I_ROWS	8	Number of rows whose \$I token_info was actually read
TRACE_QRY_I_SIZE	9	Number of bytes read from \$I LOBs
TRACE_QRY_R_TIME	10	Time spent fetching and reading \$R information
TRACE_QRY_CON_TIME	11	Time spent in CONTAINS processing (drexrcontains/drexrstart/drexrfetch)
TRACE_QRY_S_TIME	15	Time spent fetching and reading \$S information
TRACE_QRY_O_TIME	19	Time spent reading \$0 information
TRACE_QRY_D_TIME	23	Time spent reading \$D information
TRACE_QRY_SNIPPET_TIME	25	Time spent extracting a snippet from a document
TRACE_HIL_DOCSERV_TIME	26	Time spent by document service procedures (snippet, highlight, and markup)

Tracing is independent of logging. Logging does not have to be on to start tracing, and viceversa.

Traces are associated with a session—they can measure operations that take place within a single session, and conversely, cannot make measurements across sessions.

During parallel sync or optimize, the trace profile will be copied to the worker sessions only if tracing is currently enabled. Each worker will accumulate its own traces and implicitly write all trace values to the worker logfile before termination.

Related Topics

REMOVE_TRACE

GET_TRACE_VALUE

LOG_TRACES

RESET_TRACE

Oracle Text Application Developer's Guide

11.3 DISABLE_QUERY_STATS

Disables gathering of query stats for the index.

Syntax

```
ctx_output.disable_query_stats(
index_name IN VARCHAR2
);
```

index_name

The name of the index on which query stats collection is to be disabled.

Example

Turn off gathering of query stats for the index myindex.

```
CTX OUTPUT.DISABLE QUERY STATS(myindex);
```

Notes

Once the query stats is disabled for an index, gathering and storing query-related metadata is stopped for that index. All the entries corresponding to that index are cleared from the dictionary tables. An error is returned if you call this procedure on an index where query stats is not enabled.

Related Topics

```
CTX_OUTPUT."ENABLE_QUERY_STATS"
CTX_REPORT."INDEX_STATS"
```

11.4 ENABLE_QUERY_STATS

Enables gathering of query stats for the index. To have query-related metadata stored for the index, use this procedure to enable collection of statistics on that index. You can only access the gathered metadata when ctx output.enable query stats is turned on for the index.



Accessing the query stats metadata only works when ctx_output.enable_query_stats is turned on for the index. Please see
CTX_REPORT."INDEX_STATS" for the list of gathered query stats metadata.

Syntax

```
ctx_output.enable_query_stats(
index_name IN VARCHAR2
);
```

index name

The name of the index on which query stats collection is to be enabled.

Example

Turn on gathering of query stats for the index myindex.

```
CTX_OUTPUT.ENABLE_QUERY_STATS(myindex);
```

Notes

The information that shows whether query stats is enabled on an index is available in the views: CTX_INDEXES and CTX_USER_INDEXES under the column idx_query_stats_enabled, which is in both of these views. If query_stats is enabled for an index, then the column displays YES; if not, then the column displays NO.



The data corresponding to the query statistics will be stored in persistent dictionary tables. Once statistics has been enabled for a particular index, query statistics will be collected for that index from all sessions.

If you call this procedure for an index where query stats is already enabled, then an error is thrown.

Statistics collection has a minimal effect on query performance.

Related Topics

```
CTX_OUTPUT."DISABLE_QUERY_STATS"
CTX_REPORT."INDEX_STATS".
```

11.5 END_LOG

This procedure halts logging index and document service requests.

Syntax

```
ctx_output.end_log;
```

Example

```
begin
CTX_OUTPUT.END_LOG;
end:
```

ctx_output.end_query_log;

11.6 END_QUERY_LOG

Use this procedure to stop logging queries into the database trace files.

Syntax

11.7 GET_TRACE_VALUE

Use this procedure to programmatically retrieve the current value of a trace.

Syntax

```
CTX OUTPUT.GET TRACE VALUE(trace id BINARY INTEGER);
```

trace id

Specify the trace ID whose value you want. See Table 11-1 for possible values.



Example

This sets the value of the variable value:

```
value := ctx_output.get_trace_value(trace_id);
```

Notes

You can also retrieve trace values through SQL:

```
select * from ctx_trace_values;
```

See "CTX_TRACE_VALUES" for the entries in the CTX TRACE VALUES view.

If the trace has not been enabled, an error is raised.

Traces are not reset to 0 by this call.

Traces are associated with a session—they can measure operations that take place within a single session, and conversely, cannot make measurements across sessions.

Related Topics

```
"REMOVE_TRACE"

"ADD_TRACE"

"LOG_TRACES"

"RESET_TRACE"
```

Oracle Text Application Developer's Guide

11.8 LOG_TRACES

Use this procedure to print all active traces to the RDBMS trace files.

Syntax

```
CTX_OUTPUT.LOG_TRACES;
```

Notes

Traces are not reset to 0 by this call.

The traces now go to the database trace files.

Related Topics

```
"REMOVE_TRACE"
"GET_TRACE_VALUE"
"ADD_TRACE"
"RESET_TRACE"
```

Oracle Text Application Developer's Guide



11.9 LOGFILENAME

Returns the current session's trace file name. An error occurs if logging is not started.

Syntax

```
CTX OUTPUT.LOGFILENAME RETURN VARCHAR2;
```

Returns

Log file name

Example

```
declare
   logname varchar2(100);
begin
   logname := CTX_OUTPUT.LOGFILENAME;
   dbms_output.put_line('The current log file is: '||logname);
end;
```

11.10 REMOVE EVENT

Use this procedure to remove an event added through ctx output.add event.

Syntax

```
CTX_OUTPUT.REMOVE_EVENT(event in NUMBER);
```

event

Specify the type of index event to remove from the log. You can remove the following events:

- CTX_OUTPUT.EVENT_INDEX_PRINT_ROWID, which logs the rowid of each row after it is indexed. This is useful for debugging a failed index operation.
- CTX OUTPUT.EVENT OPT PRINT TOKEN, which prints each token as it is being optimized.
- CTX_OUTPUT.EVENT_INDEX_PRINT_TOKEN, which prints the each token as it is being indexed.

Example

```
begin
CTX_OUTPUT.REMOVE_EVENT(CTX_OUTPUT.EVENT_INDEX_PRINT_ROWID);
end;
```

Related Topics

```
"ADD_EVENT"
```

11.11 REMOVE_TRACE

Use this procedure to disable a trace.

Syntax

CTX_OUTPUT.REMOVE_TRACE(trace_id BINARY_INTEGER);

trace_id

Specify the ID of the trace to disable. See Table 11-1 for possible values.

Notes

If the trace has not been enabled, an error is raised.

Related Topics

```
"GET_TRACE_VALUE"
"ADD_TRACE"
"LOG_TRACES"
"RESET_TRACE"
```

Oracle Text Application Developer's Guide

11.12 RESET_TRACE

Use this procedure to clear a trace (that is, reset it to 0).

Syntax

```
CTX_OUTPUT.RESET_TRACE(trace_id BINARY_INTEGER);
```

trace id

Specify the ID of the trace to reset. See Table 11-1 for possible values.

Notes

If the trace has not been enabled, an error is raised.

Related Topics

```
"REMOVE_TRACE"
"GET_TRACE_VALUE"
"ADD_TRACE"
"LOG_TRACES"
```

Oracle Text Application Developer's Guide

11.13 START LOG

Begin logging index and document service requests. The index logs are written to the database trace files.

Syntax

CTX OUTPUT.START LOG(logfile in varchar2, overwrite in default true);

logfile

Specify the name of the log file. Starting with Oracle Database 12c Release 2 (12.2), the logfile parameter is ignored. The logs are now appended to the database trace files. Use the dictionary views such as V\$DIAG_INFO and V\$PROCESS to find the path to your current session's trace file or to the trace file for each Oracle Database process.

The Automatic Diagnostic Repository Command Interpreter (ADRCI) utility can also be used to access the trace files.

overwrite

Specify whether you want to overwrite or append to the original query log file specified by *logfile*, if it already exists. Starting with Oracle Database 12c Release 2 (12.2), this parameter is ignored. By default, all logs are appended to the database trace file.

Examples

```
begin
CTX_OUTPUT.START_LOG('mylog1');
end;
```

To view the indexing logs, search for COMPONENT_NAME='CONTEXT_INDEX' in view V\$DIAG TRACE FILE CONTENTS:

```
select PAYLOAD from V$DIAG_TRACE_FILE_CONTENTS where
COMPONENT NAME='CONTEXT INDEX' and TRACE FILENAME = trc name;
```

To view the query logs, search for COMPONENT_NAME='CONTEXT_QUERY' in view V\$DIAG TRACE FILE CONTENTS:

```
select PAYLOAD from V$DIAG_TRACE_FILE_CONTENTS where
COMPONENT NAME='CONTEXT QUERY' and TRACE FILENAME = trc name;
```

Parallel Query (PQ) workers have trace filenames of the type: SID_pxxx_PID.trc. To see the traces in the parallel workers:

```
select TRACE_FILENAME, PAYLOAD from V$DIAG_TRACE_FILE_CONTENTS where
COMPONENT NAME='CONTEXT INDEX' and TRACE FILENAME LIKE '%p00%';
```

Notes

No logs are written if the PDB lockdown profile CTX LOGGING is enabled.

Logging does not have to be on to start tracing, and vice-versa.

Logging is associated with a session-it can log operations that take place within a single session, and, conversely, cannot make measurements across sessions.

Filenames used in CTX_OUTPUT.START_LOG are restricted to the following characters: alphanumeric, minus, period, space, hash, underscore, single and double quotes. Any other character in the filename will raise an error.

11.14 START QUERY LOG

Begin logging query requests. Starting with Oracle Database 12c Release 2 (12.2), the query logs are written to the database trace files.

Use CTX_OUTPUT.END_QUERY_LOG to stop logging queries. Use CTX_REPORT.QUERY_LOG_SUMMARY to obtain reports on logged queries, such as which queries returned successfully the most times.

The query log includes the query string, the index name, and the timestamp of the query, as well as whether or not the query successfully returned a hit. A successful query for the phrase *Blues Guitarists* made at 6:46 (local time) on November 11th, 2003, would be entered into the query log in this form:

```
<QuerySet><TimeStamp>18:46:51 02/04/03</TimeStamp><IndexName>
IDX_SEARCH_TABLE</IndexName><Query>Blues
Guitarists</Query>ReturnHit>Yes</ReturnHit></QuerySet>
```

Syntax

```
CTX_OUTPUT.START_QUERY_LOG(logfile in varchar2, overwrite in default true);
```

logfile

Specify the name of the query log file. Starting with Oracle Database 12c Release 2 (12.2), the logfile parameter is ignored. The logs are appended to the database trace files instead. Use the dictionary views such as V\$DIAG_INFO and V\$PROCESS to find the path to your current session's trace file or to the trace file for each Oracle Database process.

The Automatic Diagnostic Repository Command Interpreter (ADRCI) utility can also be used to access the trace files.

overwrite

Specify whether you want to overwrite or append to the original query log file specified by logfile, if it already exists. Starting with Oracle Database 12c Release 2 (12.2), this parameter is ignored. By default, all logs are appended to the database trace file.

Example

Notes

No logs are written if the PDB lockdown profile CTX LOGGING is enabled.

Filenames used in CTX_OUTPUT.START_QUERY_LOG are restricted to the following characters: alphanumeric, minus, period, space, hash, underscore, single and double quotes. Any other character in the filename will raise an error.

Logging is associated with a session-it can log operations that take place within a single session, and, conversely, cannot make measurements across sessions.

CTX_QUERY Package

This chapter describes the CTX_QUERY PL/SQL package you can use for generating query feedback, counting hits, and creating stored query expressions.

The CTX QUERY package includes the following procedures and functions:

Name	Description
BROWSE_WORDS	Returns the words around a seed word in the index.
COUNT_HITS	Returns the number hits to a query.
EXPLAIN	Generates query expression parse and expansion information.
HFEEDBACK	Generates hierarchical query feedback information (broader term, narrower term, and related term).
REMOVE_SQE	Removes a specified stored query expression from the SQL tables.
RESULT_SET	Executes a query and generates a result set.
RESULT_SET_CLOB_QUERY	Executes a query and generates a result set based on a CLOB query parameter.
RESULT_SET_DOCUMENT	Holds the result set document after the CONTAINS query cursor is explicitly closed and if the query template has the <ctx_result_set_descriptor> element.</ctx_result_set_descriptor>
STORE_SQE	Executes a query and stores the results in stored query expression tables.
WARM_CACHE	Improves query performance by limiting disk reads from Oracle Text secondary tables.

Note:

You can use this package only when your index type is CONTEXT. This package does not support the CTXCAT index type.

The APIs in the CTX_QUERY package do not support identifiers that are prefixed with the schema or the owner name.

12.1 BROWSE_WORDS

This procedure enables you to browse words in an Oracle Text index. Specify a seed word and BROWSE_WORDS returns the words around it in the index, and an approximate count of the number of documents that contain each word.

This feature is useful for refining queries. You can identify the following words:

- Unselective words (words that have low document count)
- Misspelled words in the document set

Syntax 1: To Store Results in Table

```
ctx_query.browse_words(

index_name IN VARCHAR2,
seed IN VARCHAR2,
restab IN VARCHAR2,
browse_id IN NUMBER DEFAULT 0,
numwords IN NUMBER DEFAULT 10,
direction IN VARCHAR2 DEFAULT BROWSE_AROUND,
part_name IN VARCHAR2 DEFAULT NULL
);
```

Syntax 2: To Store Results in Memory

```
ctx_query.browse_words(

index_name IN VARCHAR2,
seed IN VARCHAR2,
resarr IN OUT BROWSE_TAB,
numwords IN NUMBER DEFAULT 10,
direction IN VARCHAR2 DEFAULT BROWSE_AROUND,
part_name IN VARCHAR2 DEFAULT NULL
);
```

index

Specify the name of the index. You can specify schema.name. Must be a local index.

seed

Specify the seed word. This word is lexed before browse expansion. The word need not exist in the token table. seed must be a single word. Using multiple words as the seed will result in an error.

restab

Specify the name of the result table. You can enter restab as schema.name. The table must exist before you call this procedure, and you must have INSERT permissions on the table. This table must have the following schema.

Column	Datatype
browse_id	number
word	varchar2(64)
doc_count	number

Existing rows in restab are not deleted before BROWSE WORDS is called.

resarr

Specify the name of the result array. resarr is of type ctx query.browse tab.

```
type browse_rec is record (
   word varchar2(64),
   doc_count number
);
type browse tab is table of browse rec index by binary integer;
```



browse id

Specify a numeric identifier between 0 and 2³². The rows produced for this browse have a value of in the <code>browse_id</code> column in <code>restab</code>. When you do not specify <code>browse_id</code>, the default is 0.

numwords

Specify the number of words returned.

direction

Specify the direction for the browse. You can specify one of:

value	behavior
BEFORE	Browse seed word and words alphabetically before the seed.
AROUND	Browse seed word and words alphabetically before and after the seed.
AFTER	Browse seed word and words alphabetically after the seed.

Symbols CTX_QUERY.BROWSE_BEFORE, CTX_QUERY.BROWSE_AROUND, and CTX_QUERY.BROWSE_AFTER are defined for these literal values as well.

part_name

Specify the name of the index partition to browse.

Example

Browsing Words with Result Table

Browsing Words with Result Array

```
set serveroutput on;
declare
  resarr ctx_query.browse_tab;
begin
ctx_query.browse_words('myindex','dog',resarr,5,CTX_QUERY.BROWSE_AROUND);
for i in 1..resarr.count loop
  dbms_output.put_line(resarr(i).word || ':' || resarr(i).doc_count);
end loop;
end;
```

12.2 COUNT HITS

Returns the number of hits for the specified query. You can call <code>COUNT_HITS</code> in exact or estimate mode. Exact mode returns the exact number of hits for the query. Estimate mode returns an upper-bound estimate but runs faster than exact mode.

Syntax

Syntax 1

Syntax 2

index name

Specify the index name.

text_query

Specify the query.

exact

Specify TRUE for an exact count. Specify FALSE for an upper-bound estimate. Specifying FALSE returns a less accurate number but runs faster. Specifying FALSE might return a number which is too high if rows have been updated or deleted since the last FULL index optimize. Optimizing in full mode removes these false hits, and then EXACT set to FALSE will return the same number as EXACT set to TRUE.

part_name

Specify the name of the index partition to query.

Notes

If the query contains structured criteria, then you should use SELECT COUNT(*).

If the index was created with the TRANSACTIONAL parameter, then COUNT_HITS will include pending rowids as well as those that have been synchronized.

12.3 EXPLAIN

Use CTX_QUERY.EXPLAIN to generate explain plan information for a query expression. The EXPLAIN plan provides a graphical representation of the parse tree for a Text query expression. This information is stored in a result table.

This procedure does *not* execute the query. Instead, this procedure can tell you how a query is expanded and parsed before you enter the query. This is especially useful for stem, wildcard, thesaurus, fuzzy, soundex, or about queries. Parse trees also show the following information:

- Order of execution (precedence of operators)
- ABOUT query normalization
- Query expression optimization
- Stop-word transformations



Breakdown of composite-word tokens

Knowing how Oracle Text evaluates a query is useful for refining and debugging queries. You can also design your application so that it uses the explain plan information to help users write better queries.

Syntax

Syntax 1

exec CTX QUERY.EXPLAIN(

index name

Specify the name of the index to be gueried.

text auerv

Specify the query expression to be used as criteria for selecting rows.

When you include a wildcard, fuzzy, or soundex operator in <code>text_query</code>, this procedure looks at the index tables to determine the expansion.

Wildcard, fuzzy (?), and soundex (!) expression feedback does not account for lazy deletes as in regular queries.

explain table

Specify the name of the table used to store representation of the parse tree for *text_query*. You must have at least INSERT and DELETE privileges on the table used to store the results from EXPLAIN.



"EXPLAIN Table" in Oracle Text Result Tables for more information about the structure of the explain table.

sharelevel

Specify whether explain_table is shared by multiple EXPLAIN calls. Specify 0 for exclusive use and 1 for shared use. Default is 0 (single-use).

When you specify 0, the system automatically truncates the result table before the next call to EXPLAIN.



When you specify 1 for shared use, this procedure does not truncate the result table. Only results with the same <code>explain_id</code> are updated. When no results with the same <code>explain_id</code> exist, new results are added to the <code>EXPLAIN</code> table.

explain_id

Specify a name that identifies the explain results returned by an EXPLAIN procedure when more than one EXPLAIN call uses the same shared EXPLAIN table. Default is NULL.

part_name

Specify the name of the index partition to query.

Example

Creating the Explain Table

To create an explain table called test explain for example, use the following SQL statement:

```
create table test_explain(
    explain_id varchar2(30),
    id number,
    parent_id number,
    operation varchar2(30),
    options varchar2(30),
    object_name varchar2(255),
    position number,
    cardinality number);
```

Running CTX_QUERY.EXPLAIN

To obtain the expansion of a query expression such as comp% OR ?smith, use CTX QUERY.EXPLAIN as follows:

```
ctx_query.explain(
    index_name => 'newindex',
    text_query => 'comp% OR ?smith',
    explain_table => 'test_explain',
    sharelevel => 0,
    explain id => 'Test');
```

Retrieving Data from Explain Table

To read the explain table, you can select the columns as follows:

```
select explain_id, id, parent_id, operation, options, object_name, position
from test_explain order by id;
```

The output is ordered by ID to simulate a hierarchical query:

EXPLAIN_ID	ID PAR	RENT_ID	OPERATION	OPTIONS	OBJECT_NAME	POSITION
Test	1	0	OR	NULL	NULL	1
Test	2	1	EQUIVALENCE	NULL	COMP%	1
Test	3	2	WORD	NULL	${\tt COMPTROLLER}$	1
Test	4	2	WORD	NULL	COMPUTER	2
Test	5	1	EQUIVALENCE	(?)	SMITH	2
Test	6	5	WORD	NULL	SMITH	1
Test	7	5	WORD	NULL	SMYTHE	2

Restrictions

CTX QUERY. EXPLAIN does not support the use of query templates.



You cannot use CTX QUERY. EXPLAIN with remote queries.

Related Topics

Oracle Text CONTAINS Query Operators

Stopword Transformations in Oracle Text

12.4 HFEEDBACK

In English or French, this procedure generates hierarchical query feedback information (broader term, narrower term, and related term) for the specified query.

Broader term, narrower term, and related term information is obtained from the knowledge base. However, only knowledge base terms that are also in the index are returned as query feedback information. This increases the chances that terms returned from HFEEDBACK produce hits over the currently indexed document set.

Hierarchical query feedback information is useful for suggesting other query terms to the user.

Syntax

Syntax 1

index name

Specify the name of the index for the text column to be queried.

text_query

Specify the query expression to be used as criteria for selecting rows.

feedback_table

Specify the name of the table used to store the feedback terms.



See Also:

"HFEEDBACK Table" in Oracle Text Result Tables for more information about the structure of the explain table.

sharelevel

Specify whether feedback_table is shared by multiple HFEEDBACK calls. Specify 0 for exclusive use and 1 for shared use. Default is 0 (single-use).

When you specify 0, the system automatically truncates the feedback table before the next call to HFEEDBACK.

When you specify 1 for shared use, this procedure does not truncate the feedback table. Only results with the same $feedback_id$ are updated. When no results with the same $feedback_id$ exist, new results are added to the feedback table.

feedback_id

Specify a value that identifies the feedback results returned by a call to HFEEDBACK when more than one HFEEDBACK call uses the same shared feedback table. Default is NULL.

part name

Specify the name of the index partition to query.

Example

Create HFEEDBACK Result Table

Create a result table to use with CTX QUERY. HFEEDBACK as follows:

```
CREATE TABLE restab (
feedback_id VARCHAR2(30),
id NUMBER,
parent_id NUMBER,
operation VARCHAR2(30),
options VARCHAR2(30),
object_name VARCHAR2(80),
position NUMBER,
bt_feedback ctxsys.ctx_feedback_type,
rt_feedback ctxsys.ctx_feedback_type,
nt_feedback ctxsys.ctx_feedback_type,
NESTED TABLE bt_feedback STORE AS res_nt,
NESTED TABLE nt_feedback STORE AS res_nt;
```

CTX_FEEDBACK_TYPE is a system-defined type in the CTXSYS schema.

See Also:

"HFEEDBACK Table" in Oracle Text Result Tables for more information about the structure of the HFEEDBACK table.

Call CTX_QUERY.HFEEDBACK

The following code calls the HFEEDBACK procedure with the query *computer industry*.



Select From the Result Table

The following code extracts the feedback data from the result table. It extracts broader term, narrower term, and related term feedback separately from the nested tables.

```
DECLARE
 i NUMBER;
BEGIN
 FOR frec IN (
   SELECT object_name, bt_feedback, rt_feedback, nt_feedback
   FROM restab
   WHERE feedback_id = 'query10' AND object_name IS NOT NULL
 ) LOOP
    dbms_output.put_line('Broader term feedback for ' || frec.object_name ||
':');
    i := frec.bt feedback.FIRST;
   WHILE i IS NOT NULL LOOP
     dbms output.put line(frec.bt feedback(i).text);
     i := frec.bt feedback.NEXT(i);
    END LOOP;
    dbms output.put line('Related term feedback for ' || frec.object name ||
':');
    i := frec.rt feedback.FIRST;
   WHILE i IS NOT NULL LOOP
     dbms output.put line(frec.rt feedback(i).text);
     i := frec.rt feedback.NEXT(i);
    END LOOP;
    dbms output.put line('Narrower term feedback for ' || frec.object name ||
':');
    i := frec.nt feedback.FIRST;
    WHILE i IS NOT NULL LOOP
     dbms output.put line(frec.nt feedback(i).text);
     i := frec.nt feedback.NEXT(i);
    END LOOP;
 END LOOP;
END;
```

Sample Output

The following output is for the preceding example, which queries on computer industry:

```
Broader term feedback for computer industry: hard sciences
Related term feedback for computer industry: computer networking
electronics
knowledge
library science
mathematics
optical technology
```



```
robotics
satellite technology
semiconductors and superconductors
symbolic logic
telecommunications industry
Narrower term feedback for computer industry:
AT&T Starlans
ATI Technologies, Incorporated
ActivCard
Actrade International Ltd.
Alta Technology
Amiga Format
Amiga Library Services
Amiga Shopper
Amstrat Action
Apple Computer, Incorporated
```



The HFEEDBACK information you obtain depends on the contents of your index and knowledge base and as such might differ from the sample shown.

Restrictions

CTX QUERY. HFEEDBACK does not support the use of query templates and rolling upgrades.

12.5 REMOVE_SQE

The CTX QUERY.REMOVE SQE procedure removes the specified stored query expression.

CTX_QUERY.REMOVE_SQE can be used to remove both session-duration and persistent SQEs. See "STORE_SQE".

Since the <code>query_name</code> namespace is shared between the persistent and session-duration SQEs, it is unnecessary to specify the duration of the SQE to be removed.

Syntax

```
CTX_QUERY.REMOVE_SQE(
          query_name IN VARCHAR2
);
```

query_name

Specify the name of the stored or session-duration query expression to be removed.

Example

```
begin
  ctx_query.remove_sqe('dis1');
  ctx_query.remove_sqe('dis2');
end;
/
```



12.6 RESULT SET

This procedure executes an XML or JSON query and generates a result set in XML or JSON.

The Result Set Interface can return data views that are difficult to express in SQL.



Oracle Text Application Developer's Guide for details on how to use the Result Set Interface

Syntax

index_name

Specify the index against which to execute the query.

query

Specify the query string.

result_set_descriptor

Specify the result set descriptor in XML or JSON. It describes what the result set should contain.

result_set

Specify the output result set. If this variable is NULL on input, a session-duration temporary lob will be allocated and returned to the user. The user is responsible for deallocating this temporary lob.

part name

Specify the index partition name. If the index is global, <code>part_name</code> must be <code>NULL</code>. If the index is partitioned and <code>part_name</code> is not <code>NULL</code>, then the query will only be evaluated for the given partition. If the index is partitioned and <code>part_name</code> is <code>NULL</code>, then the query will be evaluated for all partitions.

format

Specify the format for the result set descriptor. Use $\texttt{CTX}_Q\texttt{UERY}.XML_FORMAT$ for XML format and $\texttt{CTX}_Q\texttt{UERY}.JSON_FORMAT$ for JSON format. The default is $\texttt{CTX}_Q\texttt{UERY}.XML_FORMAT$.

The Input Result Set Descriptor

The result set descriptor is an XML message or JSON object which describes what to calculate for the result set. The elements present in the result set descriptor and the order in which they occur serve as a simple template, specifying what to include in the output result set. That is,

there should be the list of hit rowids, then a count, then a token count, and so on. The attributes of the elements specify the parameters and options to the specific operations, such as number of hits in the list of rowids, estimate versus exact count, and so on.

The XML Format Input Result Set Descriptor

The result set descriptor itself is XML conforming to the following DTD:

```
<!DOCTYPE ctx result set descriptor [</pre>
<!ELEMENT ctx result set descriptor (hitlist?, group*, count?, collocates?)>
<!ELEMENT hitlist (rowid?, score?, sdata*, snippet*, sentiment?)>
<!ELEMENT group (count?, group values?)>
<!ELEMENT count EMPTY>
<!ELEMENT rowid EMPTY>
<!ELEMENT score EMPTY>
<!ELEMENT sdata EMPTY>
<!ELEMENT group values (value*)>
<!ELEMENT value EMPTY>
<!ELEMENT sentiment (item*)>
<!ELEMENT item EMPTY>
<!ELEMENT collocates EMPTY>
<!ATTLIST sentiment classifier CDATA "DEFAULT CLASSIFIER">
<!ATTLIST item topic CDATA #REQUIRED>
<!ATTLIST item type (about | exact">
<!ATTLIST item agg (TRUE|FALSE) "FALSE">
<!ATTLIST item radius CDATA "50">
<!ATTLIST item max inst CDATA "5">
<!ATTLIST item starttag CDATA #IMPLIED>
<!ATTLIST item endtag CDATA #IMPLIED>
<!ATTLIST collocates radius CDATA "20">
<!ATTLIST collocates max words CDATA "10">
<!ATTLIST collocates use tscore (TRUE|FALSE) "TRUE">
<!ATTLIST collocates use hits CDATA "10">
<!ATTLIST group sdata CDATA #REQUIRED>
<!ATTLIST group topn CDATA #IMPLIED>
<!ATTLIST group bucketby CDATA #IMPLIED>
<!ATTLIST group sortby CDATA #IMPLIED>
<!ATTLIST group order CDATA #IMPLIED>
<!ATTLIST value id CDATA #IMPLIED>
<!ATTLIST hitlist start hit num CDATA #REQUIRED>
<!ATTLIST hitlist end hit num CDATA #REQUIRED>
<!ATTLIST hitlist order CDATA #IMPLIED>
<!ATTLIST count exact (TRUE|FALSE) "FALSE">
<!ATTLIST sdata name CDATA #REQUIRED>
<!ATTLIST snippet radius CDATA #IMPLIED>
<!ATTLIST snippet max length CDATA #IMPLIED>
<!ATTLIST snippet starttag CDATA #IMPLIED>
<!ATTLIST snippet endtag CDATA #IMPLIED>
1>
```

The following is a description of the possible XML elements for the result set descriptor:

ctx result set descriptor

This is the root element for the result set descriptor. The parent element is none, as are the available attributes.

The possible child elements are:

- Zero or more hitlist elements.
- Zero or more group elements.
- At most one count element.
- group

The group element causes the generated result set to include a group breakdown. In other words, a breakdown of the results by SDATA section values. The group element is also used to obtain facet counts for faceted navigation support. The parent element is ctx result set descriptor, and the available attributes are:

- sdata

Specifies the name of the SDATA section to use for grouping. It is required.

bucketby

Determines how group values are bucketed for counting. The single attribute displays each unique facet value along with its count. Starting with Oracle Database Release 21c, the custom attribute value is also supported which displays a range of numeric facets along with their count.

topn

Restricts the maximum number of facet values that are returned. It sorts by descending group count by default. Valid attribute values are positive integers larger than zero.

sortby

Valid attribute values are value and count. Value sorts using the value themselves, as appropriate for each data type. Count (default) sorts using the counts for each group.

order

Order can be ascending or descending.

Possible child elements of group are:

- count
- range
- hitlist

The hitlist element controls inclusion of a list of hit documents. The parent element is ctx result set descriptor, and the available attributes are:

The possible attribute elements for hitlist are:

start hit num

This specifies the starting document hit to be included in the generated result set. This can be set to any positive integer less than or equal to 16000. For example, if start_hit_num is 21, then the result set will include document hits starting from the 21st document hit. This element is required.

- end hit num

This specifies the last document hit to be included in the generated result set. This can be set to any positive integer less than or equal to 48000. For example, if end hit num



is 40, then the result set will include document hits up to the 40th document hit. This element is required.

order

This is an optional attribute that specifies the order for the documents in the generated result set. The value is a list similar to a SQL ORDER BY statement, except that, instead of column names, they can either be SCORE or SDATA section names. In the following example, MYDATE and MYPRICE are the SDATA section names:

```
(order = "SCORE DESC, MYDATE, MYPRICE DESC")
```

The possible child elements for hitlist are:

- At most one rowid element.
- At most one score element.
- One or more sdata element.
- At most one snippet element.
- count

This element causes the generated result set to include a count of the number of hit documents. The parent elements are:

- ctx_result_set_descriptor
- group

The available attributes for count are:

exact

This is to estimate mode. Set to true or false. It is required, and the default is false.

The possible child elements for count are none.

rowid

This child element causes the generated result set to include rowid information for each hit. The parent element is hitlist. There are no attributes and no possible child elements.

score

This child element causes the generated result set to include score information for each hit.

- The parent element is hitlist.
- There are no available attributes, and no possible child elements.
- sdata

This child element causes the generated result set to include sdata values for each hit.

- The parent element is hitlist.
- The available attribute is name. This specifies the name of the sdata section. It is required.
- There are no child elements.
- sentiment

This element controls the inclusion of sentiment classification results for each document returned as a part of the hitlist. There can be only one sentiment element in the hitlist element.



The parent element is hitlist.

The attribute available for this element is classifier, which specifies the sentiment classifier that is used to perform sentiment analysis. If no classifier is specified, then the CTXSYS.DEFAULT_SENTIMENT_CLASSIFIER is used. If a specified classifier is not available, then an error is displayed.

• item

This element specifies keywords or concepts for which sentiment information must be fetched for the returned set of documents. Each sentiment element must contain at least one child item element. The maximum is 10 child item elements. If you specify an empty item element (without any attributes), it indicates that sentiment score for entire document must be returned.

The parent element is sentiment.

The available attributes for item are:

topic

This specifies the topic for which sentiment analysis must be performed.

type

If this attribute value is set to ABOUT, then the classifier treats the specified topic as a concept rather than a keyword. The default is EXACT.

agg

Determines whether the sentiment score must be aggregated and presented as a single score for the entire document. The possible values are TRUE or FALSE. TRUE indicates that the per text segment scores will be aggregated and text segments will not be returned in the output resultset, only the aggregated score will be returned. The default value is FALSE.

- radius

This specifies the radius of the surrounding text to be identified during sentiment classification for that keyword. The default value is 50.

max inst

This specifies how many instances of text excerpts related to the specified topic must be analyzed for sentiment classification. The default value is 5.

starttag

This specifies the starting tag for topic highlighting.

endtag

This specifies the ending tag for topic highlighting.

collocates

This element controls the generation of related keywords or concepts associated with the collection of documents retrieved by the query.

The parent element is ctx result set descriptor.

The available attributes for collocates are:

radius

This specifies the radius of the surrounding text to be identified for collocates. The default value is 20.



max words

This specifies the maximum number of collocates to return for the given query. The default value is 10.

use_tscore

This specifies whether to use T-score for scoring the collocates. The possible values are TRUE or FALSE, with the default being TRUE.

Set this attribute to TRUE to identify collocates that are common tokens. Set this attribute to FALSE to identify collocates that emphasize unique words.

The Output Result Set XML

The output result set XML is XML conforming to the following DTD:

```
<!DOCTYPE ctx result set [</pre>
<!ELEMENT ctx result set (hitlist?, groups*, count?, collocates?)>
<!ELEMENT hitlist (hit*)>
<!ELEMENT hit (rowid?, score?, snippet*, sdata*, sentiment?)>
<!ELEMENT groups (group*)>
<!ELEMENT group (count?)>
<!ELEMENT count (#PCDATA)>
<!ELEMENT rowid (#PCDATA)>
<!ELEMENT score (#PCDATA)>
<!ELEMENT snippet (segment*)>
<!ELEMENT sdata (#PCDATA)>
<!ELEMENT sentiment (item*)>
<!ELEMENT item (segment*, score*, doc?)>
<!ELEMENT segment (segment text?, segment score?)>
<!ELEMENT segment_text (#PCDATA)>
<!ELEMENT segment_score (#PCDATA)>
<!ELEMENT doc (score?)>
<!ELEMENT collocates (collocation*)>
<!ELEMENT collocation (word?, score?)>
<!ELEMENT word (#PCDATA)>
<!ATTLIST item topic CDATA #REQUIRED>
<!ATTLIST groups sdata CDATA #REQUIRED>
<!ATTLIST group value CDATA #REQUIRED>
<!ATTLIST group range CDATA #IMPLIED>
<!ATTLIST group single CDATA #IMPLIED>
<!ATTLIST sdata name CDATA #REQUIRED>
```

The following is a description of the list of possible XML elements for the output result set:

ctx result set

This is the root element for the generated result set. There are no attributes. The parent is none. The possible child elements are:

- At most one hitlist element.
- Zero or more groups elements.
- groups

This delimits the start of a group breakdown section. The parent element is ctx result set. The available attributes are:

sdata

This is the name of the sdata section used for grouping.

The possible child elements are:

Zero or more group elements.

group

This delimits the start of a GROUP BY value. The parent element is the groups element. The available attributes are:

value

This is the value of the sdata section.

The possible child elements are at most one count element.

hitlist

This delimits the start of hitlist information. The parent element is ctx_result_set, while the children are zero or more hit elements. There are no attributes.

hit

This delimits the start of the information for a particular document within a hitlist. The parent element is hitlist, and there are no available attributes. The possible child elements are:

- Zero or one rowid elements.
- Zero or one score element.
- Zero or one sdata element.
- Zero or one snippet element.
- rowid

This is the rowid of the document, so the content is the rowid of the document. The parent element is the hit element. There are no child elements, and no available attributes.

score

This is the score of the document. The parent element is the hit element. The content is the numeric score. There are no available attributes, and no possible child elements.

• sdata

This is the SDATA value or values for the document. The parent element is the hit element, and the available attribute is name, which is the name of the sdata section. There are no possible child elements available. The content is the SDATA section value, which, for DATE values, is in the format "YYYY-MM-DD HH24:MI:SS", depending upon the actual values being stored.

count

This is the document hit count. The parent element is the <code>ctx_result_set</code> element or the <code>group</code> element. It contains the numeric hit count, has no attributes, and no possible child elements.

• sentiment

This delimits the sentiment element for the hitlist document. Its child element is item and parent is hitlist. It contains no attributes in the output result set.

item

This delimits the item element for the hitlist document. Parent element is sentiment and child elements are segment, score, and doc. It has one attribute called topic.

segment

This delimits an instance of segment element in a hit. Parent element is item. Child elements are segment text and segment score. It contains no attributes.

segment text

This specifies the text segment for the given item topic. Parent element is segment. It has no child elements or attributes.

segment score

This specifies the sentiment score for the segment. Parent element is segment. It has no child elements or attributes.

score

This specifies the sentiment score for the document or for the parent item topic. When present within collocation it specifies the collocation score for the particular collocation keyword. Parent element is doc or collocation. It has no child elements or attributes

doc

This denotes the sentiment score is for the entire document. Its parent element is item and child element is score. It has no attributes.

collocates

This delimits the collocates element for the result set output. Parent element is ctx result set and child element is collocation. It has no attributes.

collocation

This denotes a single collocation. Parent element is collocates and child elements are word and score. It has no attributes.

word

This specifies the collocates token. Its parent element is collocation. It has no child elements or attributes.

Example

This call to CTX_QUERY.RESULT_SET with the specified XML result_set_descriptor will generate the following information in the form of XML:

- top 5 hits displaying, score, rowid, author SDATA section value, and pubDate SDATA section value, order by pubDate SDATA section value DESC and score DESC
- total doc hit count for the text query
- counts group by pubDate SDATA section values
- counts group by author SDATA section values

```
declare
  rs clob;
begin
  dbms_lob.createtemporary(rs, true, dbms_lob.session);
  ctx_query.result_set('docidx', 'oracle', '
  <ctx_result_set_descriptor>
    <count/>
    <hitlist_start_hit_num="1" end_hit_num="5" order="pubDate_desc, score_desc">
        <score/>
        <rowid/>
        <sdata_name="author"/>
```



```
<sdata name="pubDate"/>
   </hitlist>
   <group sdata="pubDate">
     <count/>
   </group>
   <group sdata="author">
     <count/>
   </group>
  </ctx_result_set_descriptor>
', rs);
 dbms lob.freetemporary(rs);
exception
 when others then
  dbms lob.freetemporary(rs);
  raise;
end;
```

The XML output store in the result set output clob will resemble the following:

```
<ctx_result_set>
 <hitlist>
    <hi+>
     <score>3</score><rowid>AAAPoEAABAAAMWsAAC</rowid>
     <sdata name="AUTHOR">John</sdata>
     <sdata name="PUBDATE">2001-01-03 00:00:00</sdata>
    </hit>
    <hit>
      <score>3</score><rowid>AAAPoEAABAAAMWsAAG</rowid>
     <sdata name="AUTHOR">John</sdata>
     <sdata name="PUBDATE">2001-01-03 00:00:00</sdata>
   </hit>
    <hi+>
      <score>3</score><rowid>AAAPoEAABAAAMWsAAK</rowid>
      <sdata name="AUTHOR">John</sdata>
     <sdata name="PUBDATE">2001-01-03 00:00:00</sdata>
    </hit>
    <hit>
      <score>3</score><rowid>AAAPoEAABAAAMWsAAO</rowid>
     <sdata name="AUTHOR">John</sdata>
     <sdata name="PUBDATE">2001-01-03 00:00:00</sdata>
    </hit>
    <hit>
      <score>3</score><rowid>AAAPoEAABAAAMWsAAS</rowid>
     <sdata name="AUTHOR">John</sdata>
     <sdata name="PUBDATE">2001-01-03 00:00:00</sdata>
    </hit>
  </hitlist>
  <count>100</count>
  <groups sdata="PUBDATE">
   <group value="2001-01-01 00:00:00"><count>25</count>
    <group value="2001-01-02 00:00:00"><count>50</count></group>
    <group value="2001-01-03 00:00:00"><count>25</count></group>
  </groups>
  <groups sdata="AUTHOR">
    <group value="John"><count>50</count></group>
    <group value="Mike"><count>25</count></group>
    <group value="Steve"><count>25</count></group>
  </groups>
```



```
</ctx result set>
```

The JSON Format Input Result Set Descriptor

The JSON format result set descriptor consists of \$query, \$search, and \$facet parts. You can use the JSON format result set descriptor to query context indexes and JSON search index. It is of the following format:

```
{
  "$query": <text query and filter conditions>,
  "$search": <search result specifications>,
  "$facet": <faceted result specifications>
}
```

\$query

Use query to specify a search query, the path constraints, and additional path based filter conditions. When query is specified, the query parameter of $CTX_QUERY.RESULT_SET$ procedure is ignored.

Note:

The \$query part is supported only when a JSON search index exists on the column. You can not specify the \$query part when there is an Oracle Text index.

query is a subset of Simple Oracle Document Access (SODA) filter specification, also known as a query-by-example (QBE) or simply a filter. The following clauses are only supported:

Contains Clause - A contains clause is a field followed by an object with one \$contains operator, whose value is a string. It matches a document only if a string or number in the field value matches the string operand somewhere, including in array elements. Matching is Oracle Text full-text. You can use a contains clause only in the outermost condition of a QBE. You can also have multiple contains clauses only at the top level within a \$and operator.

For example, this QBE checks for a "name" field that contains the word "doe" and an "address" field that contains the number 10 or the string "10" as a word:



Note:

* Use wildcard field steps (*) in the contains clause to include other path steps between the paths. For example:

```
address.*.name
```

* Use descendent notation (..) in the contains clause to include descendant path steps between the paths. For example:

```
address..name2
```

In this query, name2 is considered as a descendent of address and matches the address record of the table.

- * You can use a \$contains field condition only as a part of a simple \$contains query or as a part of the outermost \$and condition. You cannot use it as a part of a \$or condition or an inner \$and condition. Doing so results in an error.
- Field-Condition Clause A field-condition clause is JSON-object member whose field is not an operator and whose value is an object with one or more members, each of which is a condition-operator clause:

```
field : { condition-operator-clause ... }
```

The following condition operators are only supported:

- * **\$eq** Matches document if field value equals operand value and the operand is a JSON scalar value. Also, matches document if field value is an array object and the operand value is an element of that array.
- * **\$gt** Matches document only if field value is greater than operand value. The operand must be a JSON number or string.
- * **\$gte** Matches document only if field value is greater than or equal to operand value. The operand must be a JSON number or string.
- * **\$lt** Matches document only if field value is less than operand value. The operand must be a JSON number or string.
- * **\$Ite** Matches document only if field value is less than or equal to operand value. The operand must be a JSON number or string.

Note:

- * Wildcard field steps (*) and array steps ([and]) are not supported.
- * To support field conditions on string values, a JSON search index with search_on_text_value_string is required.
- Logical Combining Clause A logical combining clause combines the effects of
 multiple non-empty filter conditions. A logical combining clause is a logical combining
 operator \$and or \$or followed by a non-empty array of one or more non-empty
 filter conditions. The values of the operator clauses can only be numbers or string
 values.



The following is an example of a \$query part with the supported clauses:

\$search

Use \$search to display the score ranked search results and their count. For a non-JSON Oracle Text full-text index, you can also specify the SDATA sections to project for the search results.

You can use the following attributes:

- start and end Specify the range of the search result. For example, for start = 1 and end = 10, the first 10 documents are returned.
- project Specify the list of SDATA sections to project for the search results. This
 attribute is supported only for a non-JSON Oracle Text full-text index.
- \$facet

Use \$facet to specify the facets for various paths of a JSON document or SDATA sections of a context indexed document. Facets bucketed by a single unique value and facets per user specified range buckets are supported. The facets can also be one of the aggregations like COUNT, MIN, etc.

You can specify a facet object in the following ways:

 A field as a string or numeric value for which the output has facet group counts for each single unique value of the specified field:

```
{ "$uniqueCount": { "path/sdata" : field, "type" (Optional) : "string/number" } }
```

where:

- * field refers to a SODA path for querying using a JSON search index when you use path and SDATA section name for querying using a context index when you use sdata.
- * type is either string (default) or number. When you are using sdata, the type parameter is not allowed as each sdata already has a predefined type.
- A field only for string values when using a JSON search index where field refers to a SODA path for querying using a JSON search index:

```
{ "$uniqueCount": field }
```



A field for computing aggregations on facet groups using bucket ranges:

where:

- * \$op is one of \$sum, \$min, \$max, \$avg, or \$count.
- * field refers to a SODA path for querying using a JSON search index when you use path and SDATA section name for querying using a context index when you use sdata.
- * Each range bucket must have at-most one lower bound (\$gt or \$gte) and upper bound (\$lt or \$lte).
- * type is either number (default) or string. When you are using sdata, the type parameter is not allowed as each sdata already has a predefined type.

Note:

sum and avg aggregations are only supported when the value of type parameter is number or sdata is of number type. You can only use scount, min, and max for string type.

A field only for computing aggregations on numeric facets without using bucket ranges:

```
{ "$op" : <field> }
```

\$op is one of \$sum, \$min, \$max, \$avg, or \$count.

The following is an example for \$facet part:



```
}
}
},
{ "$uniqueCount" : "book.author" },
{ "$uniqueCount " : { "path" : "book.rating", "type" :
"number" } },
{ "$avg" : "book.sales" }
{ "$min" : "book.name", "type" : "string" }
}
```

This example generates the following:

- Sum of prices for each bucket range of the specified book.price
- Total number of authors in the given specified ranges
- A group count of every unique value of book.author
- A group count of every book.rating treating the rating as a number
- Average of the book sales for all the books that satisfied the query
- The author's name that is lexicographically smallest



To support facets on string values, a JSON search index with search_on text value string is required.

The JSON Format Result Set Output

The JSON format result set output is a JSON object that consists of the following parts:

```
"$count" : number
"$hit" : [ <hit_object_1>, ..., <hit_object_i> , ... ]
"$facet": [ <facet object 1>, ..., <facet object i>, ...]
```

The following is a description of the list of possible JSON objects for the output result set:

\$count

The \$count JSON object shows the total number of hits for the query.

\$hit

The \$hit JSON object shows an array of search hit objects sorted in descending order of search score depending on how many hits were specified using start and end in the \$search part of the input query. It has the following attributes:

score

The score attribute shows the score information for each hit.

- rowid

The rowid attribute shows the rowid information for each hit.

- project



The project attribute shows the sdata values that were specified in the \$search part of the input query. The project attribute is supported only for a non-JSON Oracle Text full-text index.

• \$facet

The \$facet JSON object shows an array of facet responses for every facet specified in the \$facet part of the input query.

For enumerating counts for each unique input string or numeric value, the output is of the following format:

```
{ "<field>" : [ ..., { "value" : <value_i>, "$uniqueCount" : <group count i>}, ... ]}
```

For enumerating counts for the buckets specified in input to compute aggregations for facet groups, the output is of the following format:

```
{ "<field>" : [ ..., { "bucket" : <bucket_object_i>, "<op>" : <group count i>}, ... ]}
```

Note:

For bucket outputs, if either lower bound (\$gt or \$gte) or upper bound (\$te or \$te) are not specified in the input, then the minimum or maximum value is discovered and displayed in the output.

For enumerating counts for computing aggregations on numeric facets without using buckets, the output is of the following format:

```
{ "<field>" : { "<op>" : <actual value of the aggregation> } }
```

Example 12-1 Using the JSON format Result Set Interface with CONTEXT Index

This example shows you how to use the JSON format result set interface with CONTEXT index.

Create a table and populate it with values:



Create a section group named <code>mysecgrp</code> and enable the <code>optimized_for search</code> attribute for each column to be treated as a facet:

```
exec ctx ddl.drop section group ('mysecgrp')
exec ctx ddl.create section group ('mysecgrp', 'BASIC SECTION GROUP')
exec ctx ddl.add sdata section ('mysecgrp', 'rating', 'rating', 'NUMBER')
exec ctx ddl.set section attribute('mysecgrp', 'rating', 'optimized for',
'search')
exec ctx ddl.add sdata section ('mysecgrp', 'price', 'price', 'NUMBER')
exec ctx ddl.set section attribute('mysecgrp', 'price', 'optimized for',
'search')
exec ctx ddl.add sdata section ('mysecgrp', 'name', 'name', 'VARCHAR2')
exec ctx ddl.set section attribute('mysecqrp', 'name', 'optimized for',
'search')
exec ctx ddl.add sdata section ('mysecgrp', 'stripes', 'stripes',
'VARCHAR2')
exec ctx ddl.set section attribute('mysecgrp', 'stripes', 'optimized for',
'search')
exec ctx ddl.add sdata section ('mysecgrp', 'handler', 'handler',
'VARCHAR2')
exec ctx ddl.set section attribute('mysecgrp', 'handler', 'optimized for',
'search')
                                 ('mysecgrp', 'sold', 'sold', 'VARCHAR2')
exec ctx ddl.add sdata section
exec ctx ddl.set section attribute('mysecgrp', 'sold', 'optimized for',
'search')
```

Create a CONTEXT index on details and specify the preferences by using the parameters clause:

```
create index zebra_idx on zebra_table(details)
indextype is ctxsys.context
parameters('section group mysecgrp');
```

A call to CTX_QUERY.RESULT_SET with the specified JSON result_set_descriptor generates the following information in the form of JSON:

- · Rowids, names, and handlers for the first two hits
- Total number of unique zebra names
- Total number of sold and unsold zebras
- Total number of zebras according to their prices
- Sum of prices and average rating between a price range for the total hits and unique counts based on the sum of prices and average rating
- Total number of zebras grouped by their handler name within certain ranges

```
variable rs output clob;
declare
  gry varchar2 (4000);
  rs descriptor clob;
begin
  qry := 'zebra details';
  rs descriptor := '
  "$search" : { "start" : 1, "end" : 2, "project" : [ "name", "handler" ] },
  "$facet" : [
                    { "$uniqueCount" : "name" },
                    { "$uniqueCount" : "sold" },
                    { "$uniqueCount" : { "sdata" : "price" } },
                    { "$sum" : { "sdata" : "price",
                                 "bucket" :
                                   [ { "$lt" : 3000 }, { "$gte" : 3000 } ]
                    { "$avg" : "rating" },
                      "$count" : { "sdata" : "handler",
                                   "bucket" :
                                     [ { "$lte" : "C" }, { "$gt" : "C" } ]
                    }
             ]
}
  dbms lob.createtemporary( :rs output, true );
  ctx query.result set( 'zebra idx', qry, rs descriptor, :rs output,
                         format => CTX QUERY.JSON FORMAT );
end;
/
select json query(:rs output, '$' pretty) from dual;
The following is output:
  "$count" : 4,
  "$hit" :
```

```
"score" : 3,
   "rowid" : "AAASxXAABAAAY95AAA",
    "project" :
  "NAME" : "Storm",
  "HANDLER" : "Bob"
  },
   "score" : 3,
   "rowid" : "AAASxXAABAAAY95AAB",
   "project" :
  "NAME" : "Snowy",
  "HANDLER" : "Jane Doe"
  }
],
"$facet" :
[
   "NAME" :
   "value" : "Zigs",
   "$uniqueCount" : 2
  },
   "value" : "Snowy",
   "$uniqueCount" : 1
  },
   "value" : "Storm",
   "$uniqueCount" : 1
  },
   "SOLD" :
   "value" : "true",
    "$uniqueCount" : 2
  } ,
   "value" : "false",
   "$uniqueCount" : 1
   ]
  },
   "PRICE" :
    [
  {
```

```
"value" : 1000,
 "$uniqueCount" : 2
},
 "value" : 3000,
 "$uniqueCount" : 2
},
 "value" : 2000,
 "$uniqueCount" : 1
},
 "PRICE" :
 "bucket" :
   "$gte" : 1000,
  "$1t" : 3000
 "$sum" : 4000
},
 "bucket" :
  "$gte" : 3000,
   "$1te" : 3000
 } ,
 "$sum" : 6000
 ]
 "RATING" :
}
},
 "HANDLER" :
  "bucket" :
  "$gte" : "Bob",
   "$1te" : "C"
 "$count" : 1
},
  "bucket" :
   "$gt" : "C",
   "$lte" : "Jane Doe"
```

```
},
    "$count" : 3
}

]

}
```

Example 12-2 Using the JSON format Result Set Interface with JSON Search Index

This example shows you how to use the JSON format result set interface with JSON search index.

Create a table and populate it with values:

```
drop table zebra table;
create table zebra table(id number, details clob check(details is json));
INSERT INTO zebra table
VALUES (1, '{ "zebra" : { "price" : [2000,1000],
                       "name" : "Storm",
                       "stripes" : ["Dark", "Light"],
                       "handler" : "Bob", "sold" : true }}');
INSERT INTO zebra table
VALUES (2, '{ "zebra" : { "rating": 5, "price" : 1000,
                       "name" : "Zigzag",
                       "stripes" : ["Light", "Grey"],
                       "handler" : "Jane Doe", "sold" : "true" }}');
INSERT INTO zebra table
VALUES (3, '{ "zebra" : { "rating": 4.5, "price" : 3000,
                       "name" : "Zigs",
                       "stripes" : ["Grey", "Dark"],
                        "handler" : "Jane Doe", "sold" : false }}');
INSERT INTO zebra table
VALUES (4,'{ "zebra" : { "rating": "4.5", "price" : "3000",
                       "name" : "Zigs",
                       "stripes" : ["Grey", "Dark"],
                       "handler" : "Jane Doe", "sold" : null }}');
```

Create a JSON search index on details and specify the preferences by using the parameters clause:

```
create search index zebra_idx on zebra_table(details) for json
parameters('search on text value string');
```

A call to CTX_QUERY.RESULT_SET with the specified JSON result_set_descriptor generates the following information in the form of JSON:

- Total number of zebras that have names which satisfy the given condition
- Rowids for the first two hits that have names which satisfy the given condition
- Total number of unique zebra names

- Total number of sold and unsold zebras
- Total number of zebras according to their prices
- Sum of prices and average rating between a price range for the total hits and unique counts based on the sum of prices and average rating
- Total number of zebras grouped by their handler name within certain ranges

```
variable rs output clob;
declare
  rs descriptor clob;
begin
  rs descriptor := '
  "$query" : { "zebra.*.name" : { "$contains" : "sto% or zig%" } },
  "$search" : { "start" : 1, "end" : 2 },
  "$facet" : [
                    { "$uniqueCount" : "zebra.name" },
                    { "$uniqueCount" : "zebra.sold" },
                    { "$uniqueCount" :
                        { "path" : "zebra.price", "type" : "number" }
                   },
                    { "$sum" : { "path" : "zebra.price",
                                 "bucket" :
                                   [ { "$lt" : 3000 }, { "$gte" : 3000 } ]
                     "$avg" : "zebra.rating" },
                     "$count" : { "path" : "zebra.handler",
                                   "type" : "string",
                                   "bucket" :
                                     [ { "$lte" : "C" }, { "$gt" : "C" } ]
                   }
             ]
}
  dbms lob.createtemporary( :rs output, true );
  ctx query.result set( 'zebra idx', null, rs descriptor, :rs output,
                        format => CTX QUERY.JSON FORMAT );
end;
/
select json query(:rs output, '$' pretty) from dual;
The following is output:
  "$count" : 4,
  "$hit" :
      "score" : 4,
      "rowid": "AAASwtAABAAAY95AAB"
```

```
"score" : 4,
   "rowid" : "AAASwtAABAAAY95AAC"
],
"$facet" :
   "zebra.name" :
   [
   "value" : "Zigs",
   "$uniqueCount" : 2
   "value" : "Zigzag",
    "$uniqueCount" : 1
  },
   "value" : "Storm",
   "$uniqueCount" : 1
   ]
    "zebra.sold" :
   "value" : "true",
    "$uniqueCount" : 2
  },
   "value" : "null",
    "$uniqueCount" : 1
  },
   "value" : "false",
   "$uniqueCount" : 1
   "zebra.price" :
    "value" : 1000,
   "$uniqueCount" : 2
  },
   "value" : 3000,
    "$uniqueCount" : 2
```

```
"value" : 2000,
"$uniqueCount" : 1
},
 "zebra.price" :
 "bucket" :
   "$gte" : 1000,
  "$1t" : 3000
 "$sum" : 4000
},
 "bucket" :
  "$gte" : 3000,
 "$1te" : 3000
 } ,
 "$sum" : 6000
 "zebra.rating" :
}
},
{
 "zebra.handler" :
 "bucket" :
   "$gte" : "Bob",
   "$lte" : "C"
 } ,
 "$count" : 1
},
 "bucket" :
  "$gt" : "C",
  "$lte" : "Jane Doe"
 "$count" : 3
```

```
}
```

Limitations and Restrictions

The following limitations and restrictions apply for RESULT SET.

- The Result Set Interface (RSI) is not supported with Virtual Private Database. (VPD is supported with the regular CONTAINS query, but not with RSI.)
- In order to execute the function, you must be able to guery the base table.
- If a VPD policy is active on the base table, the documents portion of the result set will not show any documents to which you are not entitled.
- When a VPD policy is being used, aggregate measures such as count may not be accurate.

See Also:

- Oracle Text Application Developer's Guide for information on the XML and JSON Result Set Interfaces
- Oracle Text Application Developer's Guide for more information on faceted navigation
- Oracle Database Introduction to Simple Oracle Document Access (SODA) for more information on SODA filter specifications

12.7 RESULT_SET_CLOB_QUERY

This procedure executes an XML or JSON query and generates a result set based on a $\tt CLOB$ query parameter in XML or JSON.

The RESULT_SET_CLOB_QUERY procedure is identical to the RESULT_SET procedure except that the datatype of its query parameter is CLOB instead of VARCHAR2 to handle longer queries.

Syntax

See Also:

RESULT SET for the description of these parameters

12.8 RESULT SET DOCUMENT

RESULT_SET_DOCUMENT holds the result set document after the CONTAINS query cursor is explicitly closed and if the query template has the <ctx_result_set_descriptor> element.

Syntax

index name

Specify the index against which to execute the query.

query

Specify the query string.

result_set_descriptor

Specify the result set descriptor in XML or JSON. It describes what the result set should contain.

result_set

Specify the output result set. If this variable is \mathtt{NULL} on input, a session-duration temporary lob will be allocated and returned to the user. The user is responsible for deallocating this temporary lob.

part name

Specify the index partition name. If the index is global, <code>part_name</code> must be <code>NULL</code>. If the index is partitioned and <code>part_name</code> is not <code>NULL</code>, then the query will only be evaluated for the given partition. If the index is partitioned and <code>part_name</code> is <code>NULL</code>, then the query will be evaluated for all partitions.

Related Topics

"RESULT_SET"

12.9 STORE_SQE

This procedure creates either a stored or session-duration query expression (SQE). Only the query definition is stored.

SQEs are used to store the definition of a query without storing any results. Referencing the query with the CONTAINS SQL operator references the definition of the query. In this way, SQEs make it easy for defining long or frequently used query expressions. Creating a session-duration SQE is useful for when you do not want the maintenance overhead of deleting unused or no longer needed SQEs.

Supported Operators

Stored query expressions support all of the CONTAINS query operators. Stored query expressions also support all of the special characters and other components that can be used in a query expression, including other stored query expressions.

Privileges

Users are permitted to create and remove stored query expressions owned by them. Users are permitted to use stored query expressions owned by anyone. The CTXSYS user can create or remove stored query expressions for any user.

Syntax

Syntax 1

query_name

Specify the name of the stored guery expression to be created.

text_query

Specify the query expression to be associated with query name.

duration

The possible values are DURATION SESSION and DURATION PERSISTENT.

- When duration is to set to DURATION_SESSION, the stored query expression is stored in a
 PL/SQL package variable and is available for the session.
- When duration is to set to DURATION_PERSISTENT, the stored query expression is stored in a database table, and can be referenced by other database sessions.
- SQEs with the DURATION_SESSION option are not supported when issued from the catalog
 of a sharded database. Use the DURATION PERSISTENT option instead.
- The query_name namespace is shared between the persistent and session-duration SQEs. If you try to add a persistent or session-duration SQE with a name that is already used by another persistent or session-duration SQE, then an error will be raised.

duration_persistent

When there is a CLOB query, specify that the duration is stored in a database table. This SQE must be deleted when it is no longer needed.

The query_name namespace is shared between the persistent and session-duration SQEs. If you try to add a persistent or session-duration SQE with a name that is already used by another persistent or session-duration SQE, then an error will be raised.

Example

```
begin
  ctx_query.store_sqe('dis1', 'flood', CTX_QUERY.DURATION_SESSION);
  ctx_query.store_sqe('dis2', 'tornado', CTX_QUERY.DURATION_PERSISTENT);
  ctx query.store_sqe('dis3', 'fire')
```



```
end;
```

Restriction

SQEs are supported in logical standby only starting with Oracle Database release 23ai and not in previous releases of Oracle Database.

12.10 WARM CACHE

The CTX_QUERY.WARM_CACHE PL/SQL procedure improves query performance by limiting disk reads from Oracle Text secondary tables.

To run this operation, ensure that you specify a large buffer cache by using the buffer cache initialization parameters.

Syntax

```
CTX_QUERY.WARM_CACHE(

owner_name IN VARCHAR2,
index_name IN

VARCHAR2,
table_identifier IN VARCHAR2 default
'I',
part_name IN VARCHAR2 default null
);
```

owner_name

The name of the owner of the index for which you want to improve the query performance.

index name

The name of the index.

table identifier

The name of the Oracle Text secondary table that you want to load.

part name

If your index is a local index, then you must specify the name of the index partition to synchronize otherwise an error is returned. If your index is a global, nonpartitioned index, then specify NULL, which is the default.

Example

```
CTX_QUERY.WARM_CACHE('owner_1', 'IDX', 'I', 'p1');
```

Related Topics

Oracle Database Administrator's Guide

CTX_REPORT Package

You can use the CTX_REPORT package to create reports on indexing and querying. These reports can help you troubleshoot problems or fine-tune your applications.

- Description of Procedures in CTX_REPORT
- Using the Function Versions
- DESCRIBE_INDEX
- DESCRIBE_POLICY
- CREATE_INDEX_SCRIPT
- CREATE_POLICY_SCRIPT
- INDEX_SIZE
- INDEX_STATS
- QUERY_LOG_SUMMARY
- SHOW_TOKENS
- TOKEN INFO
- TOKEN_TYPE
- VALIDATE_INDEX

Note:

The APIs in the CTX_REPORT package do not support identifiers that are prefixed with the schema or the owner name.

See Also:

Oracle Text Application Developer's Guide for an overview of the CTX_REPORT package and how you can use the various procedures described in this chapter

13.1 Description of Procedures in CTX REPORT

The CTX REPORT package contains the following procedures:

Name	Description
DESCRIBE_INDEX	Creates a report describing the index.
DESCRIBE_POLICY	Creates a report describing a policy.
CREATE_INDEX_SCRIPT	Creates a SQL*Plus script to duplicate the named index.

Name	Description	
CREATE_POLICY_SCRIPT	Creates a SQL*Plus script to duplicate the named policy.	
INDEX_SIZE	Creates a report to show the internal objects of an index, their tablespaces and used sizes.	
INDEX_STATS	Creates a report to show the various statistics of an index.	
QUERY_LOG_SUMMARY	Creates a report showing query statistics	
TOKEN_INFO	Creates a report showing the information for a token, decoded.	
TOKEN_TYPE	Translates a name and returns a numeric token type.	
VALIDATE_INDEX	Checks for index corruption and reports on problems found. Mainly used with Oracle Support.	

13.2 Using the Function Versions

Some of the procedures in the CTX_REPORT package have function versions. You can call these functions as follows:

```
select ctx_report.describe_index('MYINDEX') from dual;
```

In SQL*Plus, to generate an output file to send to support, you can do:

```
set long 64000
set pages 0
set heading off
set feedback off
spool outputfile
select ctx_report.describe_index('MYINDEX') from dual;
spool off
```

13.3 DESCRIBE_INDEX

Creates a report describing the index. This includes the settings of the index metadata, the indexing objects used, the settings of the attributes of the objects, and index partition descriptions, if any.

You can call this operation as a procedure with an IN OUT CLOB parameter or as a function that returns the report as a CLOB.

Syntax

index_name

Specify the name of the index to describe.

report

Specify the CLOB locator to which to write the report.

If report is NULL, a session-duration temporary CLOB will be created and returned. It is the caller's responsibility to free this temporary CLOB as needed.

The report CLOB will be truncated before report is generated, so any existing contents will be overwritten by this call.

report format

Specify whether the report should be generated as 'TEXT' or as 'XML'. TEXT is the default. You can also specify the values CTX REPORT.FMT TEXT OF CTX REPORT.FMT XML.

Notes

CTX_REPORT.DESCRIBE_INDEX outputs FILTER BY and ORDER BY column information if the index is created with FILTER BY and/or ORDER BY clauses.

Related Topics

"CREATE INDEX"

"ADD_SDATA_COLUMN"

13.4 DESCRIBE POLICY

Creates a report describing the policy. This includes the settings of the policy metadata, the indexing objects used, and the settings of the attributes of the objects.

You can call this operation as a procedure with an IN OUT CLOB parameter or as a function that returns the report as a CLOB.

Syntax

report

Specify the CLOB locator to which to write the report.

If report is NULL, a session-duration temporary CLOB will be created and returned. It is the caller's responsibility to free this temporary CLOB as needed.

The report CLOB will be truncated before report is generated, so any existing contents will be overwritten by this call.

report_format

Specify whether the report should be generated as 'TEXT' or as 'XML'. TEXT is the default. You can also specify the values CTX REPORT.FMT TEXT OF CTX REPORT.FMT XML.

policy_name

Specify the name of the policy to describe.



13.5 CREATE_INDEX_SCRIPT

Creates a SQL*Plus script which will create a text index that duplicates the named text index.

The created script will include creation of preferences identical to those used in the named text index. However, the names of the preferences will be different.

You can call this operation as a procedure with an IN OUT CLOB parameter or as a function that returns the report as a CLOB.

Syntax

index name

Specify the name of the index.

report

Specify the CLOB locator to which to write the script.

If report is NULL, a session-duration temporary CLOB will be created and returned. It is the caller's responsibility to free this temporary CLOB as needed.

The report CLOB will be truncated before report is generated, so any existing contents will be overwritten by this call.

prefname prefix

Specify optional prefix to use for preference names.

If prefname_prefix is omitted or NULL, index name will be used. The prefname_prefix follows index length restrictions.

Notes

CTX_REPORT.CREATE_INDEX_SCRIPT will also generate necessary FILTER BY and ORDER BY clauses for CREATE INDEX statements.

Related Topics

"CREATE INDEX"

13.6 CREATE_POLICY_SCRIPT

Creates a SQL*Plus script which will create a text policy that duplicates the named text policy.

The created script will include creation of preferences identical to those used in the named text policy.

You can call this operation as a procedure with an IN OUT CLOB parameter or as a function that returns the report as a CLOB.

Syntax

policy_name

Specify the name of the policy.

report

Specify the locator to which to write the script.

If report is NULL, a session-duration temporary CLOB will be created and returned. It is the caller's responsibility to free this temporary CLOB as needed.

The report CLOB will be truncated before report is generated, so any existing contents will be overwritten by this call.

prefname_prefix

Specify the optional prefix to use for preference names. If prefname_prefix is omitted or NULL, policy name will be used. prefname prefix follows policy length restrictions.

13.7 INDEX_SIZE

Creates a report showing the internal objects of the text index or text index partition, and their tablespaces, allocated, and used sizes.

You can call this operation as a procedure with an IN OUT CLOB parameter, or as a function that returns the report as a CLOB.

Syntax

index name

Specify the name of the index to describe.

report

Specify the CLOB locator to which to write the report.

If report is NULL, a session-duration temporary CLOB will be created and returned. It is the caller's responsibility to free this temporary CLOB as needed.



The report CLOB will be truncated before report is generated, so any existing contents will be overwritten by this call

part name

Specify the name of the index partition (optional). If part_name is NULL, and the index is a local partitioned text index, then all objects of all partitions will be displayed. If part_name is provided, then only the objects of a particular partition will be displayed.

report_format

Specify whether the report should be generated as 'TEXT' or as 'XML'. TEXT is the default. You can also specify the values CTX REPORT.FMT TEXT or CTX REPORT.FMT XML.

Notes

CTX REPORT.INDEX SIZE will also output information on dr\$indexname\$S table.

Related Topics

"CREATE INDEX"

Table 2-39

13.8 INDEX_STATS

Creates a report showing various calculated statistics about the text index.

This procedure fully scans the text index tables, so it may take a long time to run for large indexes.

Syntax

index name

Specify the name of the index to describe. This must be a CONTEXT index.

report

Specify the CLOB locator to which to write the report. If report is NULL, a session-duration temporary CLOB will be created and returned. It is the caller's responsibility to free this temporary CLOB as needed.

The report CLOB will be truncated before report is generated, so any existing contents will be overwritten by this call.

part_name

Specify the name of the index partition. If the index is a local partitioned index, then part_name must be provided. INDEX STATS will calculate the statistics for that index partition.



frag_stats

Specify TRUE to calculate fragmentation statistics. If <code>frag_stats</code> is FALSE, the report will not show any statistics relating to size of index data. However, the operation should take less time and resources to calculate the token statistics.

list size

Specify the number of elements in each compiled list. <code>list_size</code> has a maximum value of 1000.

report_format

Specify whether the report should be generated as 'TEXT' or as 'XML'. TEXT is the default. You can also specify the values CTX REPORT.FMT TEXT or CTX REPORT.FMT XML.

stat_type

Specify the estimated statistics to output. If this parameter is set, then frag_stats is ignored. The possible values are:

Statistics Type	Description
EST_FRAG_STATS	Get the estimated fragmentation stats for the index. When this type is given, list_size is ignored.
EST_FREQUENT_TOKENS	Get the estimated frequently queried tokens for the index. You can give a value of up to 100 for list_size.
EST_TOKENS_TO_OPTIMIZE	Show best tokens to optimize, based on frequency of querying and fragmentation. You can give a value of up to 100 for list_size.
EST_SLOWEST_QUERIES	Show slowest running queries for the index. You can give a value of up to 100 for list_size.

Note:

The estimated statistics for stat_type is only available if query_stats is enabled and the following privileges must be granted to the user running the report:

```
grant select, insert, delete, update on ctxsys.dr$slowqrys to <user>;
grant select, insert, delete, update on ctxsys.dr$freqtoks to <user>;
```

Example for CTX_REPORT.INDEX_STATS

```
create table output (result CLOB);

declare
    x clob := null;
begin
    ctx_report.index_stats('tdrbprx21',x);
    insert into output values (x);
    commit;
    dbms_lob.freetemporary(x);
end;
/
set long 32000
```



```
set head off
set pagesize 10000
select * from output;
```

The following sample output is for INDEX_STATS on a context index. This report has been truncated for clarity. It shows some of the token statistics and all of the fragmentation statistics.

The fragmentation statistics are at the end of the report. It tells you optimal row fragmentation, an estimated amount of garbage data in the index, and a list of the most fragmented tokens. Running $CTX_DDL.OPTIMIZE_INDEX$ cleans up the index.

STATISTICS FOR "DR_TEST"."TDRBPRX21"	
indexed documents:	53
allocated docids:	68
\$I rows:	16 , 259
MOVIN CHARTON	
TOKEN STATISTICS	
unique tokens:	13,445
average \$I rows for each token:	1.21
tokens with most \$I rows:	
telecommunications industry (THEME)	6
science and technology (THEME)	6
EMAIL (FIELD SECTION "SOURCE")	6
DEC (FIELD SECTION "TIMESTAMP")	6
electronic mail (THEME) computer networking (THEME)	6 6
communications (THEME)	6
95 (FIELD SECTION "TIMESTAMP")	6
15 (FIELD SECTION "TIMESTAMP")	6
HEADLINE (ZONE SECTION)	6
average size for each token:	8
tokens with largest size:	
T (NORMAL)	405
SAID (NORMAL)	313
HEADLINE (ZONE SECTION)	272
NEW (NORMAL)	267
I (NORMAL)	230
MILLION (PREFIX)	222
D (NORMAL)	219
MILLION (NORMAL) U (NORMAL)	215 192
DEC (FIELD SECTION "TIMESTAMP")	192
average frequency for each token:	2.00
most frequent tokens:	
HEADLINE (ZONE SECTION)	68
DEC (FIELD SECTION "TIMESTAMP")	62
95 (FIELD SECTION "TIMESTAMP")	62
15 (FIELD SECTION "TIMESTAMP")	62
T (NORMAL)	61
D (NORMAL)	59
881115 (THEME)	58
881115 (NORMAL)	58
I (NORMAL)	55
geography (THEME)	52



token statistics by type: token type: unique tokens: total rows: average rows: total size: average size: average frequency: most frequent tokens:	NORMAL 6,344 7,631 1.20 67,445 (65.86 KB) 11 2.33
T D 881115 I SAID C NEW MILLION FIRST COMPANY	61 59 58 55 45 43 36 32 28 27
<pre>token type: unique tokens: total rows: average rows: total size: average size: average frequency: most frequent tokens:</pre>	THEME 4,563 5,523 1.21 21,930 (21.42 KB) 5 2.40
881115 political geography geography United States business and economics abstract ideas and concepts North America science and technology NKS nulls	58 52 52 51 50 48 48 46 34

The fragmentation portion of this report is as follows:

FRAGMENTATION STATISTICS total size of \$I data: 116,772 (114.04 KB) \$I rows: 16,259 estimated \$I rows if optimal: 13,445 estimated row fragmentation: 17 % garbage docids: 15 estimated garbage size: 21,379 (20.88 KB) most fragmented tokens: telecommunications industry (THEME) 83 % science and technology (THEME) 83 % EMAIL (FIELD SECTION "SOURCE") 83 % DEC (FIELD SECTION "TIMESTAMP") 83 % electronic mail (THEME) 83 % computer networking (THEME) 83 %



```
communications (THEME) 83 % 95 (FIELD SECTION "TIMESTAMP") 83 % HEADLINE (ZONE SECTION) 83 % 15 (FIELD SECTION "TIMESTAMP") 83 %
```

Examples for CTX_REPORT.INDEX_STATS with STAT_TYPE

The following sample output is for EST FRAG STATS statistics type:

```
var report clob;
   begin
       dbms lob.createtemporary(:report, true);
       ctx report.index stats(
       index name => 'tdrbps.idx',
               => :report,
       report
       report format => 'XML',
       stat type => 'EST FRAG STATS'
   end;
select :report from dual;
:REPORT
<CTXREPORT>
<INDEX STATS>
<STAT INDEX NAME>"TDRBPS"."IDX"</STAT INDEX NAME>
<STAT INDEX STATS>
<STAT STATISTIC NAME="Estimated Fragmentation Stats">50</STAT STATISTIC>
</STAT INDEX STATS>
</INDEX STATS>
</CTXREPORT>
```

The following sample output is for EST FREQUENT TOKENS statistics type:

```
<INDEX STATS>
<STAT INDEX NAME>"TDRBPS"."IDX"</STAT INDEX NAME>
<STAT INDEX STATS>
<STAT TOKEN LIST NAME="Most Frequently Queried Tokens">
<STAT TOKEN>
<STAT TOKEN TEXT>ORACLE</STAT TOKEN TEXT>
<STAT TOKEN TYPE>0:TEXT</STAT TOKEN TYPE>
<STAT TOKEN STATISTIC>2</STAT TOKEN STATISTIC>
</STAT TOKEN>
<STAT TOKEN>
<STAT TOKEN_TEXT>DATABASE</STAT_TOKEN_TEXT>
<STAT_TOKEN_TYPE>0:TEXT</STAT_TOKEN_TYPE>
<STAT TOKEN STATISTIC>1</STAT TOKEN STATISTIC>
</STAT TOKEN>
</STAT TOKEN LIST>
</STAT INDEX STATS>
</INDEX STATS>
</CTXREPORT>
```

Note:

<STAT_TOKEN_STATISTIC> shows the number of times a particular token was queried.

The following sample output is for EST SLOWEST QUERIES statistics type:

```
begin
        dbms lob.createtemporary(:report, true);
        ctx report.index stats(
        index_name => 'tdrbps.idx',
report => :report,
        report format => 'XML',
        stat type => 'EST SLOWEST QUERIES'
        );
    end;
select :report from dual;
:REPORT
<CTXREPORT>
<INDEX STATS>
<STAT INDEX NAME>"TDRBPS"."IDX"</STAT INDEX NAME>
<STAT INDEX STATS>
<STAT QUERY LIST NAME="Slowest Queries">
<STAT QUERY>
<STAT QUERY FULL>select count(*) from tbl where
contains(txt,'Oracle')>0</STAT QUERY FULL>
<STAT QUERY TEXT PART>Oracle</STAT QUERY TEXT PART>
<STAT QUERY TIME>114</STAT QUERY TIME>
```

```
<STAT QUERY HASH>2992140927</STAT QUERY HASH>
</STAT QUERY>
<STAT QUERY>
<STAT QUERY FULL>select count(*) from tbl where
contains(txt,'ora%')>0</STAT QUERY FULL>
<STAT QUERY TEXT PART>ora%</STAT QUERY TEXT PART>
<STAT QUERY TIME>4</STAT QUERY TIME>
<STAT QUERY HASH>2229259029</STAT QUERY HASH>
</STAT QUERY>
<STAT QUERY>
<STAT QUERY FULL>select count(*) from tbl where
contains(txt,'Database')>0</STAT_QUERY_FULL>
<STAT QUERY TEXT PART>Database/STAT QUERY TEXT PART>
<STAT QUERY TIME>2</STAT QUERY TIME>
<STAT QUERY HASH>1111113040</STAT QUERY HASH>
</STAT QUERY>
</stat_QUERY LIST>
</STAT INDEX STATS>
</INDEX STATS>
</CTXREPORT>
```

Note:

- <STAT_QUERY_FULL> contains the full query and <STAT_QUERY_TEXT_PART>
 contains the Oracle Text CONTAINS clause of the query.
- <STAT_QUERY_TIME> contains query response times and <STAT_QUERY_HASH> contains the hash values of the queries.

The following sample output is for EST TOKENS TO OPTIMIZE statistics type:

```
begin
       dbms lob.createtemporary(:report, true);
       ctx report.index stats(
       index name => 'tdrbps.idx',
               => :report,
       report
       report_format => 'XML',
       stat type => 'EST TOKENS TO OPTIMIZE'
   end;
select :report from dual;
:REPORT
<CTXREPORT>
<INDEX STATS>
<STAT INDEX NAME>"TDRBPS"."IDX"</STAT INDEX NAME><STAT INDEX STATS>
<STAT TOKEN LIST NAME="Best Tokens To Optimize">
<STAT TOKEN>
```

```
<STAT_TOKEN_TEXT>ORACLE</STAT_TOKEN_TEXT>

<STAT_TOKEN_TYPE>0:TEXT</STAT_TOKEN_TYPE>

<STAT_TOKEN_STATISTIC>100</STAT_TOKEN_STATISTIC>

</STAT_TOKEN>

<STAT_TOKEN>

<STAT_TOKEN_TEXT>DATABASE</STAT_TOKEN_TEXT>

<STAT_TOKEN_TYPE>0:TEXT</STAT_TOKEN_TYPE>

<STAT_TOKEN_STATISTIC>50</STAT_TOKEN_STATISTIC>

</STAT_TOKEN_LIST>

</STAT_TOKEN_LIST>

</STAT_INDEX_STATS>

</CTXREPORT>
```

Note:

<STAT TOKEN STATISTIC> indicates the fragmentation of a particular token.

Notes

These metadata are available only when QUERY_STATS is turned on for the index: estimated fragmentation stats, estimated frequently queried tokens, estimated most fragmented frequently queried token, and estimated slowest running queries for the specified index.

 $\verb|CTX_REPORT.INDEX_STATS| will also output information on dr$| indexname S table, which is the section data, or SDATA, table.$

Related Topics

- ENABLE_QUERY_STATS
- DISABLE_QUERY_STATS
- CREATE INDEX

Use the CREATE INDEX statement to create an Oracle Text index.

BASIC STORAGE Attributes

The BASIC_STORAGE indexing type supports these attributes for database tables and indexes.

13.9 QUERY LOG SUMMARY

Obtain a report of logged queries.

QUERY_LOG_SUMMARY enables you to analyze queries you have logged. For example, suppose you have an application that searches a database of large animals, and your analysis of queries against it shows that users are continually searching for the word *mouse*; this analysis might induce you to rewrite your application so that a search for *mouse* redirects the user to a database for small animals instead of simply returning an unsuccessful search.

With query analysis, you can find out the following:

- Which queries were made
- Which queries were successful
- Which queries were unsuccessful



How many times each query was made

You can combine these factors in various ways, such as determining the 50 most frequent unsuccessful queries made by your application.

Query logging is begun with CTX_OUTPUT.START_QUERY_LOG and terminated with CTX_OUTPUT.END QUERY_LOG.



You must connect as CTXSYS to use CTX REPORT.QUERY LOG SUMMARY.

See Also:

"START_QUERY_LOG" and "END_QUERY_LOG"

Syntax

logfile

Specify the name of the logfile that contains the queries. Starting with Oracle Database 12c release 2 (12.2), this parameter is ignored as all the query logs are written to database trace files.

indexname

Specify the name of the context index for which you want the summary report. If you specify NULL, the procedure provides a summary report for all context indexes.

result table

Specify the name of the in-memory table of type TABLE OF RECORD where the results of the QUERY_LOG_SUMMARY are to go. The default is the location specified by the system parameter LOG DIRECTORY.

row_num

The number of rows of results from QUERY_LOG_SUMMARY to be reported into the table named by restab. For example, if this is number is 10, most_freq is TRUE, and has_hit is TRUE, then the procedure returns the 10 most frequent queries that were successful (that is, returned hits).

most_freq

Specify whether <code>QUERY_LOG_SUMMARY</code> should return the most frequent or least frequent queries. The default is most frequent queries. If <code>most_freq</code> is set to <code>FALSE</code>, the procedure returns the least successful queries.

has hit

Specify whether QUERY_LOG_SUMMARY should return queries that are successful (that is, that generate hits) or unsuccessful queries. The default is to count successful queries; set has_hit to FALSE to return unsuccessful queries.

Example

The following example shows how a query log can be used.

First connect as CTXSYS. Then create and populate two tables, and then create an index for each:

```
create table qlogtab1 (tk number primary key, text varchar2(2000));
insert into qlogtab1 values(1, 'The Roman name for France was Gaul.');
insert into qlogtab1 values(2, 'The Tour de France is held each summer.');
insert into qlogtab1 values(3, 'Jacques Anatole Thibault took the pen name Anatole France.');
create index idx_qlog1 on qlogtab1(text) indextype is ctxsys.context;
create table qlogtab2 (tk number primary key, text varchar2(2000));
insert into qlogtab2 values(1, 'The Great Wall of China is about 2400 kilometers long');
insert into qlogtab2 values(2, 'Soccer dates back at least to 217 C.E.');
insert into qlogtab2 values(3, 'The Corn Palace is a tourist attraction in South Dakota.');
create index idx_qlog2 on qlogtab2(text) indextype is ctxsys.context;
```

Turn on guery logging, creating a log called guery log:

```
exec ctx_output.start_query_log('query.log');
```

Now make some queries (some of which will be unsuccessful):

```
select text from qlogtab1 where contains(text, 'France',1)>0;
select text from qlogtab1 where contains(text, 'cheese',1)>0;
select text from glogtab1 where contains(text, 'Text Wizard',1)>0;
select text from glogtab2 where contains(text, 'Corn Palace',1)>0;
select text from qlogtab2 where contains(text, 'China',1)>0;
select text from qlogtab1 where contains(text, 'Text Wizards',1)>0;
select text from qlogtab2 where contains(text, 'South Dakota',1)>0;
select text from qlogtabl where contains(text, 'Text Wizard',1)>0;
select text from qlogtab2 where contains(text, 'China',1)>0;
select text from qlogtab1 where contains(text, 'Text Wizard',1)>0;
select text from qlogtab2 where contains(text, 'company',1)>0;
select text from qlogtab1 where contains(text, 'Text Wizard',1)>0;
select text from qlogtab1 where contains(text, 'France',1)>0;
select text from glogtabl where contains(text, 'database',1)>0;
select text from qlogtab2 where contains(text, 'high-tech',1)>0;
select text from qlogtab1 where contains(text, 'database',1)>0;
select text from qlogtab1 where contains(text, 'France',1)>0;
select text from qlogtab1 where contains(text, 'Japan',1)>0;
select text from qlogtab1 where contains(text, 'Egypt',1)>0;
select text from qlogtab1 where contains(text, 'Argentina',1)>0;
select text from qlogtab1 where contains(text, 'Argentina',1)>0;
select text from qlogtab1 where contains(text, 'Argentina',1)>0;
select text from qlogtab1 where contains(text, 'Japan',1)>0;
select text from glogtab1 where contains(text, 'Egypt',1)>0;
select text from glogtab1 where contains(text, 'Air Shuttle',1)>0;
select text from glogtab1 where contains(text, 'Argentina',1)>0;
```

With the querying over, turn query logging off:

```
exec ctx_output.end_query_log;
```

Use QUERY_LOG_SUMMARY to get query reports. In the first instance, you ask to see the three most frequent queries that return successfully. First declare the results table (the queries).

This returns the following:

```
The 3 most frequent queries returning hits number of times query string
3 France
2 China
1 Corn Palace
```

Next, look for the three most frequent queries on idx qlog1 that were successful.

Because only the queries for *France* were successful, ctx_report.query_log_summary returns the following:

```
The 3 most frequent queries returning hits for index idx_qlog1
number of times query string
3 France
```

Lastly, ask to see the three least frequent queries that returned no hits (that is, queries that were unsuccessful and called infrequently). In this case, you are interested in queries on both context indexes, so you set the indexname parameter to NULL.

```
end;
```

This returns the following results:

```
The 3 least frequent queries returning no hit number of times query string

1 high-tech

1 company

1 cheese
```

Argentina and Japan do not make this list, because they are queried more than once, while Corn Palace does not make this list because it is successfully queried.

13.10 SHOW_TOKENS

The CTX_REPORT.SHOW_TOKENS function creates a report showing all the tokens (that is, words) in the main index (\$I and \$G index tables).

You can call this operation as a function that returns the report as a CLOB. This API displays all the tokens in the index, ordered by token and token type. By default, all the tokens are sorted alphabetically, without any header and with each token appearing on a new line.

For example:

CAT DOG FROG HORSE

Syntax

index name

Specify the name of the index.

part_name

Specify the name of the index partition.

If the index is a local partitioned index, then part_name must be provided. SHOW_TOKENS will apply to that index partition.

report_format

Specify whether the report should be generated in a plain text or JSON format:

- FMT TXT to print the report as plain text. This is the default format.
- FMT JSON to print the report in a JSON format.



types

Specify TRUE to print the $token_type$ number and the corresponding $type_name$ of section:

- 0: TEXT
- 9: STEM
- 101: SORTABLE SDATA
- 400: MDATA

For example:

```
CAT
      0 TEXT
DOG 101 SORTABLE SDATA
FROG 101 SORTABLE SDATA
HORSE 400 MDATA
JSON Format:
[
      "token": "CAT",
      "token type":0,
      "type name":"TEXT"
   },
      "token": "DOG",
      "token type":101,
      "type name": "TEXT"
   },
      "token": "FROG",
      "token type":101,
      "type name": "SORTABLE SDATA"
   },
      "token": "HORSE",
      "token type":400,
      "type name": "MDATA"
   }
]
```

docid_counts

Specify TRUE to include the docid (document ID) count, that is, the total number of documents associated with the token.

For example:



```
{
    "token":"DOG",
    "docid_count":3
},
{
    "token":"FROG",
    "docid_count":10
},
{
    "token":"HORSE",
    "docid_count":4
}
```

docid_ranges

Specify TRUE to include the docid_first and docid_last details, that is, the number of tokens in the first and last document (within a range of documents associated with the token). For example:

```
JSON Format:
   {
      "token": "CAT",
      "docid first":2,
      "docid last":3
   },
      "token": "DOG",
      "docid first":4,
      "docid last":7
   },
      "token": "FROG",
      "docid first":10,
      "docid last":25
   },
      "token": "HORSE",
      "docid first":1,
      "docid last":25
]
```

frag_counts

Specify TRUE to include the token_count and fragment_count. Here, fragment_count refers to the number of rows in the index tables (\$I table, or \$G table, or both) used to store the specified token. Note that the fragment_count is not a direct fragmentation measure, but can be used indirectly to assess a "low" or "high" fragmentation for the token.

For example, when the token_count is 3 and the fragment_count is 3, this implies that each instance of the token got stored as a separate row (instead of one compacted row). This indicates fragmentation, which can be resolved by running index optimization.



For example:

```
JSON Format:
      "token": "CAT",
      "token type":101,
      "token count":3,
      "fragment count":1,
      "docid first":2,
      "docid last":3,
      "docid count":2
   },
   {
      "token": "DOG",
      "token type":101,
      "token count":5,
      "fragment count":2,
      "docid first":4,
      "docid last":7,
      "docid count":2
   },
      "token": "FROG",
      "token type":201,
      "token count":12,
      "fragment count":2,
      "docid first":10,
      "docid last":25,
      "docid count":12
   },
      "token": "HORSE",
      "token type":400,
      "token count":10,
      "fragment count":2,
      "docid first":1,
      "docid last":25,
      "docid count":8
]
```

Related Topics

TOKEN_TYPE

This is a helper function which translates an English name into a numeric token type.

13.11 TOKEN_INFO

Creates a report showing the information for a token, decoded. This procedure will fully scan the info for a token, so it may take a long time to run for really large tokens.

You can call this operation as a procedure with an IN OUT CLOB parameter or as a function that returns the report as a CLOB.

Syntax

index name

Specify the name of the index.

report

Specify the CLOB locator to which to write the report.

If report is NULL, a session-duration temporary CLOB will be created and returned. It is the caller's responsibility to free this temporary CLOB as needed.

The report CLOB will be truncated before report is generated, so any existing contents will be overwritten by this call token may be case-sensitive, depending on the passed-in token type.

token

Specify the token text.

token_type

Specify the token type. You can use a number returned by the TOKEN_TYPE function. THEME, ZONE, ATTR, PATH, and PATH ATTR tokens are case-sensitive.

Everything else gets passed through the lexer, so if the index's lexer is case-sensitive, the token input is case-sensitive.

part_name

Specify the name of the index partition.

If the index is a local partitioned index, then part_name must be provided. TOKEN_INFO will apply to just that index partition.

raw info

Specify TRUE to include a hex dump of the index data. If raw_info is TRUE, the report will include a hex dump of the raw data in the token info column.

decoded info

Specify decode and include docid and offset data. If decoded_info is FALSE, CTX_REPORT will not attempt to decode the token information. This is useful when you just want a dump of data.



report_format

Specify whether the report should be generated as 'TEXT' or as 'XML'. TEXT is the default. You can also specify the values CTX_REPORT.FMT_TEXT or CTX_REPORT.FMT_XML.

13.12 TOKEN_TYPE

This is a helper function which translates an English name into a numeric token type.

This is suitable for use with token info, or any other CTX API which takes in a token type.

index_name

Specify the name of the index.

type_name

Specify an English name for token_type. The following strings are legal input. All input is case-insensitive.

Input	Meaning	Type Returned
TEXT	Normal text token	0
THEME	Theme token	1
ZONE SEC	Zone token	2
ATTR TEXT	Text that occurs in attribute	4
ATTR SEC	Attribute section	5
PREFIX	Prefix token	6
PATH SEC	Path section	7
PATH ATTR	Path attribute section	8
STEM	Stem form token	9
FIELD <name> TEXT</name>	Text token in field section <name></name>	16-79
SORTABLE SDATA	Sortable SDATA section	101
MDATA	MDATA section	400
FIELD <name> PREFIX</name>	Prefix token in field section <name></name>	616-916
FIELD <name> STEM</name>	Stem token in field section <name></name>	916-979
NDATA <name></name>	NDATA-type token	200-299

Input	Meaning	Type Returned
TOKEN_TYPE_ATTR_TXT_PFIX	Attribute text prefix	604
TOKEN_TYPE_ATTR_TXT_STEM	Attribute text stem	904

For FIELD types, the index metadata needs to be read, so if you are going to be calling this a lot for such things, you might want to consider caching the values in local variables rather than calling token_type over and over again.

The constant types (0 - 9) also have constants in this package defined.

Notes

To get token types for MDATA tokens, do not use CTX_REPORT.TOKEN_TYPE; use the MDATA operator instead. (See "MDATA".) The syntax to use is 'MDATA secname'.

Example

typenum := ctx report.token type('myindex', 'field author text');

13.13 VALIDATE_INDEX

Provides diagnostics if index corruption is believed to have occurred.

CTX_REPORT.VALIDATE_INDEX checks an index (or a partition for a locally partitioned index) and reports whether or not any corruption has been detected. VALIDATE_INDEX only checks \$I rows that have token_type 0 and does not check other rows that contain information about sections, such as the NDATA section.

This procedure is primarily intended as a diagnostic tool to be used under the direction of Oracle Support.



CTX_THES Package

This chapter provides reference information for using the CTX_THES package to manage and browse thesauri. These thesaurus functions are based on the ISO-2788 and ANSI Z39.19 standards except where noted.

Knowing how information is stored in your thesaurus helps in writing queries with thesaurus operators. You can also use a thesaurus to extend the knowledge base, which is used for ABOUT queries in English and French and for generating document themes.

CTX THES contains the following stored procedures and functions:

Name	Description
ALTER_PHRASE	Alters thesaurus phrase.
ALTER_THESAURUS	Renames or truncates a thesaurus.
ВТ	Returns all broader terms of a phrase.
BTG	Returns all broader terms generic of a phrase.
BTI	Returns all broader terms instance of a phrase.
ВТР	Returns all broader terms partitive of a phrase.
CREATE_PHRASE	Adds a phrase to the specified thesaurus.
CREATE_RELATION	Creates a relation between two phrases.
CREATE_THESAURUS	Creates the specified thesaurus.
CREATE_TRANSLATION	Creates a new translation for a phrase.
DROP_PHRASE	Removes a phrase from thesaurus.
DROP_RELATION	Removes a relation between two phrases.
DROP_THESAURUS	Drops the specified thesaurus from the thesaurus tables.
DROP_TRANSLATION	Drops a translation for a phrase.
EXPORT_THESAURUS	Exports a thesaurus from the thesaurus tables.
HAS_RELATION	Tests for the existence of a thesaurus relation.
IMPORT_THESAURUS	Imports a thesaurus into the thesaurus tables.
NT	Returns all narrower terms of a phrase.
NTG	Returns all narrower terms generic of a phrase.
NTI	Returns all narrower terms instance of a phrase.
NTP	Returns all narrower terms partitive of a phrase.
OUTPUT_STYLE	Sets the output style for the expansion functions.
PT	Returns the preferred term of a phrase.
RT	Returns the related terms of a phrase
SN	Returns scope note for phrase.
SYN	Returns the synonym terms of a phrase
THES_TT	Returns all top terms for phrase.
TR	Returns the foreign equivalent of a phrase.

Name	Description
TRSYN	Returns the foreign equivalent of a phrase, synonyms of the phrase, and foreign equivalent of the synonyms.
TT	Returns the top term of a phrase.
UPDATE_TRANSLATION	Updates an existing translation.



The APIs in the CTX_THES package do not support identifiers that are prefixed with the schema or the owner name.

✓ See Also:

Oracle Text CONTAINS Query Operators for more information about the thesaurus operators.

14.1 ALTER_PHRASE

Alters an existing phrase in the thesaurus. Only CTXSYS or thesaurus owner can alter a phrase.

Syntax

```
CTX_THES.ALTER_PHRASE(tname in varchar2, phrase in varchar2, op in varchar2, operand in varchar2 default null);
```

tname

Specify the thesaurus name.

phrase

Specify a phrase to alter.

op (alter operation)

Specify the alter operation as a string or symbol. You can specify one of the following operations with the op and operand pair:

op (or alter operation)	meaning	operand
RENAME or CTX_THES.OP_RENAME	Rename phrase. If the new phrase already exists in the thesaurus, this procedure raises an exception.	Specify a new phrase. You can include qualifiers to change, add, or remove qualifiers from phrases.
PT or CTX_THES.OP_PT	Make phrase the preferred term. Existing preferred terms in the synonym ring becomes non-preferred synonym.	(none)



op (or alter operation)	meaning	operand
SN	Change the scope note on	Specify a new scope note.
or	the phrase.	
CTX_THES.OP_SN		

operand

Specify an argument to the alter operation. See table for "op (alter operation)".

Examples

Correct misspelled word in thesaurus:

```
ctx_thes.alter_phrase('thes1', 'tee', 'rename', 'tea');
```

Remove qualifier from mercury (metal):

```
ctx_thes.alter_phrase('thes1', 'mercury (metal)', 'rename', 'mercury');
```

Add qualifier to mercury:

```
ctx_thes.alter_phrase('thes1', 'mercury', 'rename', 'mercury (planet)');
```

Make Kowalski the preferred term in its synonym ring:

```
ctx_thes.alter_phrase('thes1', 'Kowalski', 'pt');
```

Change scope note for view cameras:

```
ctx thes.alter phrase('thes1', 'view cameras', 'sn', 'Cameras with lens focusing');
```

14.2 ALTER_THESAURUS

Use this procedure to rename or truncate an existing thesaurus. Only the thesaurus owner or CTXSYS can invoke this function on a given thesaurus.

Syntax

```
CTX_THES.ALTER_THESAURUS(tname in varchar2, op in varchar2, operand in varchar2 default null);
```

tname

Specify the thesaurus name.

op

Specify the alter operation as a string or symbol. You can specify one of two operations:

ор	Meaning	operand
RENAME	Rename thesaurus. Returns an error if the new	Specify a new thesaurus name.
or		inesaurus name.
CTX_THES.OP_RENAME	name already exists.	
TRUNCATE	Truncate thesaurus.	None.
or		
CTX_THES.OP_TRUNCATE		

operand

Specify the argument to the alter operation. See table for op.



Examples

Rename thesaurus THES1 to MEDICAL:

```
ctx_thes.alter_thesaurus('thes1', 'rename', 'medical');
or
ctx thes.alter thesaurus('thes1', ctx thes.op rename, 'medical');
```

You can use symbols for any op argument, but all further examples will use strings.

Remove all phrases and relations from thesaurus THES1:

```
ctx_thes.alter_thesaurus('thes1', 'truncate');
```

14.3 BT

This function returns all broader terms of a phrase as recorded in the specified thesaurus.

Syntax 1: Table Result

Syntax 2: String Result

```
CTX_THES.BT(phrase IN VARCHAR2,

lvl IN NUMBER DEFAULT 1,

tname IN VARCHAR2 DEFAULT 'DEFAULT')

RETURN VARCHAR2;
```

restab

Optionally, specify the name of the expansion table to store the results. This table must be of type EXP TAB which the system defines as follows:

```
type exp_rec is record (
     xrel varchar2(12),
     xlevel number,
     xphrase varchar2(256)
);
type exp_tab is table of exp_rec index by binary_integer;
```

See Also:

"CTX_THES Result Tables and Data Types" in Oracle Text Result Tables for more information about $\texttt{EXP}_\texttt{TAB}$

phrase

Specify a phrase to lookup in thesaurus.

Ιv

Specify how many levels of broader terms to return. For example 2 means get the broader terms of the broader terms of the phrase.



tname

Specify a thesaurus name. If not specified, system default thesaurus is used.

Returns

This function returns a string of broader terms in the form:

```
{bt1}|{bt2}|{bt3} ...
```

Example

String Result

Consider a thesaurus named MY_THES that has an entry for cat as follows:

```
cat
BT1 feline
BT2 mammal
BT3 vertebrate
BT4 animal
```

To look up the broader terms for *cat* up to two levels, enter the following statements:

```
set serveroutput on

declare
  terms varchar2(2000);
begin
  terms := ctx_thes.bt('CAT', 2, 'MY_THES');
  dbms_output.put_line('The broader expansion for CAT is: '||terms);
end;
```

This code produces the following output:

```
The broader expansion for CAT is: {cat}|{feline}|{mammal}
```

Table Result

The following example performs a broader term lookup for brown wolf using the table result:

```
set serveroutput on

declare
   xtab ctx_thes.exp_tab;
begin
   ctx_thes.bt(xtab, 'brown wolf', 2, 'my_thesaurus');
   for i in 1..xtab.count loop
      dbms_output.put_line(xtab(i).rel||' '||xtab(i).phrase);
   end loop;
end;
```

This code produces the following output:

```
PHRASE BROWN WOLF
BT WOLF
BT CANINE
BT ANIMAL
```

Related Topics

OUTPUT_STYLE

Broader Term (BT_ BTG_ BTP_ BTI)

14.4 BTG

This function returns all broader terms generic of a phrase as recorded in the specified thesaurus.

Syntax 1: Table Result

Syntax 2: String Result

```
CTX_THES.BTG(phrase IN VARCHAR2,

lv1 IN NUMBER DEFAULT 1,

tname IN VARCHAR2 DEFAULT 'DEFAULT')

RETURN VARCHAR2;
```

restab

Optionally, specify the name of the expansion table to store the results. This table must be of type EXP TAB which the system defines as follows:

```
type exp_rec is record (
    xrel varchar2(12),
    xlevel number,
    xphrase varchar2(256)
);
type exp tab is table of exp rec index by binary integer;
```

See Also:

"CTX_THES Result Tables and Data Types" in Oracle Text Result Tables for more information about ${\tt EXP}\ {\tt TAB}.$

phrase

Specify a phrase to lookup in thesaurus.

lvl

Specify how many levels of broader terms to return. For example 2 means get the broader terms of the broader terms of the phrase.

tname

Specify thesaurus name. If not specified, system default thesaurus is used.

Returns

This function returns a string of broader terms generic in the form:

```
{bt1}|{bt2}|{bt3} ...
```

Example

To look up the broader terms generic for cat up to two levels, enter the following statements:

```
set serveroutput on declare
```



```
terms varchar2(2000);
begin
  terms := ctx_thes.btg('CAT', 2, 'MY_THES');
  dbms_output.put_line('the broader expansion for CAT is: '||terms);
end;
```

Related Topics

```
"OUTPUT_STYLE"

"Broader Term (BT_ BTG_ BTP_ BTI)"
```

14.5 BTI

This function returns all broader terms instance of a phrase as recorded in the specified thesaurus.

Syntax 1: Table Result

Syntax 2: String Result

```
CTX_THES.BTI(phrase IN VARCHAR2,

lvl IN NUMBER DEFAULT 1,

tname IN VARCHAR2 DEFAULT 'DEFAULT')

RETURN VARCHAR2;
```

restab

Optionally, specify the name of the expansion table to store the results. This table must be of type EXP TAB which the system defines as follows:

```
type exp_rec is record (
    xrel varchar2(12),
    xlevel number,
    xphrase varchar2(256)
);
type exp_tab is table of exp_rec index by binary_integer;
```

See Also:

"CTX_THES Result Tables and Data Types" in Oracle Text Result Tables for more information about $\texttt{EXP}_\texttt{TAB}$.

phrase

Specify a phrase to lookup in thesaurus.

lν

Specify how many levels of broader terms to return. For example 2 means get the broader terms of the broader terms of the phrase.

tname

Specify a thesaurus name. If not specified, system default thesaurus is used.

Returns

This function returns a string of broader terms instance in the form:

```
{bt1}|{bt2}|{bt3} ...
```

Example

To look up the broader terms instance for *cat* up to two levels, enter the following statements:

```
set serveroutput on
declare
  terms varchar2(2000);
begin
  terms := ctx_thes.bti('CAT', 2, 'MY_THES');
  dbms_output.put_line('the broader expansion for CAT is: '||terms);
end;
```

Related Topics

```
"OUTPUT_STYLE"

"Broader Term (BT_ BTG_ BTP_ BTI)"
```

14.6 BTP

This function returns all broader terms partitive of a phrase as recorded in the specified thesaurus.

Syntax 1: Table Result

Syntax 2: String Result

```
CTX_THES.BTP(phrase IN VARCHAR2,

lvl IN NUMBER DEFAULT 1,

tname IN VARCHAR2 DEFAULT 'DEFAULT')
RETURN VARCHAR2;
```

restab

Optionally, specify the name of the expansion table to store the results. This table must be of type EXP TAB which the system defines as follows:

```
type exp_rec is record (
     xrel varchar2(12),
     xlevel number,
     xphrase varchar2(256)
);
type exp_tab is table of exp_rec index by binary_integer;
```



See Also:

"CTX_THES Result Tables and Data Types" in Oracle Text Result Tables for more information about EXP TAB.

phrase

Specify a phrase to lookup in thesaurus.

lvl

Specify how many levels of broader terms to return. For example 2 means get the broader terms of the broader terms of the phrase.

tname

Specify a thesaurus name. If not specified, the system default thesaurus is used.

Returns

This function returns a string of broader terms in the form:

```
{bt1}|{bt2}|{bt3} ...
```

Example

To look up the two broader terms partitive for cat, enter the following statements:

```
declare
  terms varchar2(2000);
begin
  terms := ctx_thes.btp('CAT', 2, 'MY_THES');
  dbms_output.put_line('the broader expansion for CAT is: '||terms);
end:
```

Related Topics

```
"OUTPUT STYLE"
```

"Broader Term (BT_ BTG_ BTP_ BTI)"

14.7 CREATE_PHRASE

The CREATE PHRASE procedure adds a new phrase to the specified thesaurus.



Even though you can create thesaurus relations with this procedure, Oracle recommends that you use <code>CTX_THES.CREATE_RELATION</code> rather than <code>CTX_THES.CREATE_PHRASE</code> to create relations in a thesaurus.

Syntax

```
CTX_THES.CREATE_PHRASE(tname IN VARCHAR2, phrase IN VARCHAR2, rel IN VARCHAR2 DEFAULT NULL, relname IN VARCHAR2 DEFAULT NULL);
```



tname

Specify the name of the thesaurus in which the new phrase is added or the existing phrase is located.

phrase

Specify the phrase to be added to a thesaurus or the phrase for which a new relationship is created.

rel

Specify the new relationship between *phrase* and *relname*. This parameter is supported only for backward compatibility. Use CTX_THES.CREATE_RELATION to create new relations in a thesaurus.

relname

Specify the existing phrase that is related to *phrase*. This parameter is supported only for backward compatibility. Use CTX_THES.CREATE_RELATION to create new relations in a thesaurus.

Returns

The ID for the entry.

Example

In this example, two new phrases (os and *operating system*) are created in a thesaurus named tech thes.

```
begin
   ctx_thes.create_phrase('tech_thes','os');
   ctx_thes.create_phrase('tech_thes','operating system');
end;
```

14.8 CREATE_RELATION

Creates a relation between two phrases in the thesaurus. The synonym ring is limited in length to about 4000 synonyms, depending on word length.



Oracle recommends that you use CTX_THES.CREATE_RELATION rather than CTX_THES.CREATE_PHRASE to create relations in a thesaurus.

Only thesaurus owner and CTXSYS can invoke this procedure on a given thesaurus.

Syntax

```
CTX_THES.CREATE_RELATION(tname in varchar2, phrase in varchar2, rel in varchar2, relphrase in varchar2);
```

tname

Specify the thesaurus name

phrase

Specify the phrase to alter or create. If phrase is a disambiguated homograph, you must specify the qualifier. If phrase does not exist in the thesaurus, it is created.

rel

Specify the relation to create. The relation is from phrase to relphrase. You can specify one of the following relations:

relation	meaning	relphrase
BT*/NT*	Add hierarchical relation.	Specify the related phrase. The relationship is interpreted from phrase to relphrase.
RT	Add associative relation.	Specify the phrase to associate.
SYN	Add phrase to a synonym ring.	Specify an existing phrase in the synonym ring.
Specify language	Add translation for a phrase.	Specify a new translation phrase.

relphrase

Specify the related phrase. If relphrase does not exist in tname, relphrase is created. See table for rel.

Notes

The relation you specify for rel is interpreted as from phrase to relphrase. For example, consider dog with broader term animal:

```
dog
BT animal
```

To add this relation, specify the arguments as follows:

```
begin
CTX_THES.CREATE_RELATION('thes','dog','BT','animal');
end;
```



The order in which you specify arguments for $CTX_THES.CREATE_RELATION$ is different from the order you specify them with $CTX_THES.CREATE_PHRASE$.

Examples

Create relation VEHICLE NT CAR:

```
ctx_thes.create_relation('thes1', 'vehicle', 'NT', 'car');
```

Create Japanese translation for you:

```
ctx thes.create relation('thes1', 'you', 'JAPANESE:', 'kimi');
```

14.9 CREATE_THESAURUS

The CREATE_THESAURUS procedure creates an empty thesaurus with the specified name in the thesaurus tables.

Syntax

```
CTX_THES.CREATE_THESAURUS(name IN VARCHAR2, casesens IN BOOLEAN DEFAULT FALSE);
```

name

Specify the name of the thesaurus to be created. The name of the thesaurus must be unique. If a thesaurus with the specified name already exists, <code>CREATE_THESAURUS</code> returns an error and does not create the thesaurus.

casesens

Specify whether the thesaurus to be created is case-sensitive. If casesens is *true*, Oracle Text retains the cases of all terms entered in the specified thesaurus. As a result, queries that use the thesaurus are case-sensitive.

Example

```
begin
    ctx_thes.create_thesaurus('tech_thes', FALSE);
end;
```

14.10 CREATE_TRANSLATION

Use this procedure to create a new translation for a phrase in a specified language.

Syntax

```
CTX_THES.CREATE_TRANSLATION(tname in varchar2, phrase in varchar2, language in varchar2, translation in varchar2);
```

tname

Specify the name of the thesaurus, using no more than 30 characters.

phrase

Specify the phrase in the thesaurus to which to add a translation. Phrase must already exist in the thesaurus, or an error is raised.

language

Specify the language of the translation, using no more than 10 characters.

translation

Specify the translated term, using no more than 256 characters.

If a translation for this phrase already exists, this new translation is added without removing that original translation, so long as that original translation is not the same. Adding the same translation twice results in an error.

Example

The following code adds the Spanish translation for *dog* to *my thes*:

```
begin
    ctx_thes.create_translation('my_thes', 'dog', 'SPANISH', 'PERRO');
end;
```



14.11 DROP_PHRASE

Removes a phrase from the thesaurus. Only thesaurus owner and CTXSYS can invoke this procedure on a given thesaurus.

Syntax

```
CTX_THES.DROP_PHRASE(tname in varchar2, phrase in varchar2);
```

tname

Specify thesaurus name.

phrase

Specify a phrase to drop. If the phrase is a disambiguated homograph, then you must include the qualifier. If the phrase does not exist in tname, then this procedure raises an exception. BT* / NT* relations are patched around the dropped phrase. For example, if A has a BT B, and B has BT C, after B is dropped, A has BT C.

When a word has multiple broader terms, then a relationship is established for each narrower term to each broader term.

Note that BT, BTG, BTP, and BTI are separate hierarchies, so if A has BTG B, and B has BTI C, when B is dropped, there is no relation implicitly created between A and C.

RT relations are not patched. For example, if A has RT B, and B has RT C, then if B is dropped, there is no associative relation created between A and C.

Example

Assume you have the following relations defined in *mythes*:

```
wolf
BT canine
canine
BT animal
```

You drop phrase canine:

```
begin
ctx_thes.drop_phrase('mythes', 'canine');
end;
```

The resulting thesaurus is patched and looks like:

```
wolf
BT animal
```

14.12 DROP RELATION

Removes a relation between two phrases from the thesaurus.



 ${\tt CTX_THES.DROP_RELATION}\ removes\ only\ the\ relation\ between\ two\ phrases.\ Phrases\ are\ never\ removed\ by\ this\ call.$

Only thesaurus owner and CTXSYS can invoke this procedure on a given thesaurus.

Syntax

```
CTX_THES.DROP_RELATION(tname in varchar2, phrase in varchar2, rel in varchar2, relphrase in varchar2 default null);
```

tname

Specify the thesaurus name.

phrase

Specify the filing phrase.

rel

Specify the relation to drop. The relation is from phrase to relphrase. You can specify one of the following relations:

relation	meaning	relphrase
BT*/NT*	Remove hierarchical relation.	Optional specify relphrase. If not provided, all relations of that type for the phrase are removed.
RT	Remove associative relation.	Optionally specify relphrase. If not provided, all RT relations for the phrase are removed.
SYN	Remove phrase from its synonym ring.	(none)
PT	Remove preferred term designation from the phrase. The phrase remains in the synonym ring.	(none)
language	Remove a translation from a phrase.	Optionally specify relphrase. You can specify relphrase when there are multiple translations for a phrase for the language, and you want to remove just one translation. If relphrase is NULL, all translations for the phrase for the language are removed.

relphrase

Specify the related phrase.

Notes

The relation you specify for rel is interpreted as from phrase to relphrase. For example, consider dog with broader term animal:

```
dog
BT animal
```

To remove this relation, specify the arguments as follows:

```
begin
CTX_THES.DROP_RELATION('thes','dog','BT','animal');
end;
```



You can also remove this relation using NT as follows:

```
begin
CTX_THES.DROP_RELATION('thes', 'animal', 'NT', 'dog');
end;
```

Example

Remove relation VEHICLE NT CAR:

```
ctx_thes.drop_relation('thes1', 'vehicle', 'NT', 'car');
```

Remove all narrower term relations for vehicle:

```
ctx thes.drop relation('thes1', 'vehicle', 'NT');
```

Remove Japanese translations for me:

```
ctx_thes.drop_relation('thes1', 'me', 'JAPANESE:');
```

Remove a specific Japanese translation for *me*:

```
ctx_thes.drop_relation('thes1', 'me', 'JAPANESE:', 'boku')
```

14.13 DROP_THESAURUS

The DROP_THESAURUS procedure deletes the specified thesaurus and all of its entries from the thesaurus tables.

Syntax

```
CTX THES.DROP THESAURUS (name IN VARCHAR2);
```

name

Specify the name of the thesaurus to be dropped.

Example

```
begin
ctx_thes.drop_thesaurus('tech_thes');
end;
```

14.14 DROP_TRANSLATION

Use this procedure to remove one or more translations for a phrase.

Syntax

```
CTX_THES.DROP_TRANSLATION (tname in varchar2, phrase in varchar2, language in varchar2 default null, translation in varchar2 default null);
```

tname

Specify the name of the thesaurus, using no more than 30 characters.

phrase

Specify the phrase in the thesaurus to which to remove a translation. The phrase must already exist in the thesaurus or an error is raised.

language

Optionally, specify the language of the translation, using no more than 10 characters. If not specified, the translation must also not be specified and all translations in all languages for the phrase are removed. An error is raised if the phrase has no translations.

translation

Optionally, specify the translated term to remove, using no more than 256 characters. If no such translation exists, an error is raised.

Example

The following code removes the Spanish translation for dog:

```
begin
   ctx_thes.drop_translation('my_thes', 'dog', 'SPANISH', 'PERRO');
end;
```

To remove all translations for dog in all languages:

```
begin
    ctx_thes.drop_translation('my_thes', 'dog');
end;
```

14.15 EXPORT_THESAURUS

Use this procedure to export a thesaurus as a clob from the Oracle Text thesaurus tables. The format of the exported thesaurus is same as that of the format of the thesaurus file that is used by the ctxload utility to import thesaurus into the Oracle Text thesaurus tables.



"Thesaurus Loader (ctxload)" in Oracle Text Utilities for more information about the ctxload utility.

Only the owner of the thesaurus, or the sys user, or the ctxsys user can export a thesaurus from the Oracle Text thesaurus tables using export thesaurus.

You should call ctx_output.start_log before calling export_thesaurus to log the operations done by export thesaurus.

Syntax

name

Specify the name of the thesaurus in the Oracle Text thesaurus tables that you want to export. If the specified thesaurus does not exists in the Oracle Text thesaurus tables, then this procedure raises an exception.

thedumo

Specify the name of the clob where you want to store the thesaurus that is exported from the Oracle Text thesaurus tables.



Example

The following example copies the thesaurus named mythesaurus from the Oracle Text thesaurus tables into the clob mythesdump:

```
declare
  mythesdump clob;
begin
  ctx_thes.export_thesaurus('mythesaurus', mythesdump);
end:
```

14.16 HAS_RELATION

HAS_RELATION tests that a thesaurus relation exists without actually performing the expansion. The function returns TRUE if the phrase has any of the relations in the specified list.

Syntax

phrase

Specify the phrase.

re

Specify a single thesaural relation or a comma-delimited list of relations, except PT. Specify 'ANY' for any relation.

tname

Specify the thesaurus name.

Example

The following example returns TRUE if the phrase *cat* in the DEFAULT thesaurus has any broader terms or broader generic terms:

```
set serveroutput on
result boolean;
begin
  result := ctx_thes.has_relation('cat','BT,BTG');
  if (result) then dbms_output.put_line('TRUE');
  else dbms_output.put_line('FALSE');
  end if;
end;
```

14.17 IMPORT_THESAURUS

Use this procedure to import a thesaurus into the Oracle Text thesaurus tables. You should call <code>ctx_output.start_log</code> before calling <code>import_thesaurus</code> to log the operations done by <code>import_thesaurus</code>.

Syntax

```
CTX_THES.IMPORT_THESAURUS(name in varchar2, content in CLOB, thescase in varchar2 default 'N');
```

name

Specify the name of the thesaurus to be created. If the name of the thesaurus specified in the name parameter already exists in the Oracle Text thesaurus tables, then this procedure raises an exception.

content

Specify the thesaurus content to be imported in the Oracle Text thesaurus tables. The format of the thesaurus to be imported should be the same as used by the ctxload utility. If the format of the thesaurus to be imported is not correct, then this procedure raises an exception.



"Thesaurus Loader (ctxload)" in Oracle Text Utilities for more information about the ctxload utility.

thecase

Specify 'Y' to create a case-sensitive thesaurus and 'N' to create a case-insensitive thesaurus. The default is 'N'.

Example

The following example creates a case-sensitive thesaurus named mythesaurus and imports the thesaurus content present in myclob into the Oracle Text thesaurus tables:

```
declare
  myclob clob;
begin
  myclob := to_clob('peking SYN beijing BT capital country NT beijing tokyo');
  ctx_thes.import_thesaurus('mythesaurus', myclob, 'Y');
end;
```

14.18 NT

This function returns all narrower terms of a phrase as recorded in the specified thesaurus.

Syntax 1: Table Result

Syntax 2: String Result

```
CTX_THES.NT(phrase IN VARCHAR2,

lvl IN NUMBER DEFAULT 1,

tname IN VARCHAR2 DEFAULT 'DEFAULT')

RETURN VARCHAR2;
```



restab

Optionally, specify the name of the expansion table to store the results. This table must be of type EXP_TAB which the system defines as follows:

```
type exp_rec is record (
    xrel varchar2(12),
    xlevel number,
    xphrase varchar2(256)
);
type exp_tab is table of exp_rec index by binary_integer;
```

See Also:

"CTX_THES Result Tables and Data Types" in Oracle Text Result Tables for more information about $\texttt{EXP}_\texttt{TAB}$.

phrase

Specify a phrase to lookup in thesaurus.

ΙVΙ

Specify how many levels of narrower terms to return. For example 2 means get the narrower terms of the narrower terms of the phrase.

tname

Specify thesaurus name. If not specified, system default thesaurus is used.

Returns

This function returns a string of narrower terms in the form:

```
{nt1}|{nt2}|{nt3} ...
```

Example

String Result

Consider a thesaurus named MY THES that has an entry for cat as follows:

```
Cat
NT domestic cat
NT wild cat
BT mammal
mammal
BT animal
domestic cat
NT Persian cat
NT Siamese cat
```

To look up the narrower terms for *cat* down to two levels, enter the following statements:

```
declare
  terms varchar2(2000);
begin
  terms := ctx_thes.nt('CAT', 2, 'MY_THES');
  dbms_output.put_line('the narrower expansion for CAT is: '||terms);
end;
```

This code produces the following output:

```
the narrower expansion for CAT is: \{cat\} | \{domestic\ cat\} | \{Persian\ cat\} | \{Siamese\ cat\} | \{wild\ cat\}
```

Table Result

The following code does an narrower term lookup for canine using the table result:

```
declare
  xtab ctx_thes.exp_tab;
begin
  ctx_thes.nt(xtab, 'canine', 2, 'my_thesaurus');
  for i in 1..xtab.count loop
   dbms_output.put_line(lpad(' ', 2*xtab(i).xlevel) ||
    xtab(i).xrel || ' ' || xtab(i).xphrase);
  end loop;
end;
```

This code produces the following output:

```
PHRASE CANINE

NT WOLF (Canis lupus)

NT BROWN WOLF

NT GREY WOLF

NT DOG (Canis familiaris)

NT PIT BULL

NT DASCHUND

NT CHIHUAHUA

NT HYENA (Canis mesomelas)

NT COYOTE (Canis latrans)
```

Related Topics

OUTPUT_STYLE

Narrower Term (NT_ NTG_ NTP_ NTI)

14.19 NTG

This function returns all narrower terms generic of a phrase as recorded in the specified thesaurus.

Syntax 1: Table Result

Syntax 2: String Result

```
CTX_THES.NTG(phrase IN VARCHAR2,

lvl IN NUMBER DEFAULT 1,

tname IN VARCHAR2 DEFAULT 'DEFAULT')

RETURN VARCHAR2;
```

restab

Optionally, specify the name of the expansion table to store the results. This table must be of type EXP TAB which the system defines as follows:

```
type exp_rec is record (
    xrel varchar2(12),
    xlevel number,
    xphrase varchar2(256)
);
type exp tab is table of exp rec index by binary integer;
```



"CTX_THES Result Tables and Data Types" in Oracle Text Result Tables for more information about EXP TAB.

phrase

Specify a phrase to lookup in thesaurus.

lvl

Specify how many levels of narrower terms to return. For example 2 means get the narrower terms of the narrower terms of the phrase.

tname

Specify the thesaurus name. If not specified, system default thesaurus is used.

Returns

This function returns a string of narrower terms generic in the form:

```
{nt1}|{nt2}|{nt3} ...
```

Example

To look up the narrower terms generic for *cat* down to two levels, enter the following statements:

```
declare
  terms varchar2(2000);
begin
  terms := ctx_thes.ntg('CAT', 2, 'MY_THES');
  dbms_output.put_line('the narrower expansion for CAT is: '||terms);
end;
```

Related Topics

```
"OUTPUT_STYLE"
```

"Narrower Term (NT_ NTG_ NTP_ NTI)"

14.20 NTI

This function returns all narrower terms instance of a phrase as recorded in the specified thesaurus.

Syntax 1: Table Result



Syntax 2: String Result

```
CTX_THES.NTI(phrase IN VARCHAR2,

lvl IN NUMBER DEFAULT 1,

tname IN VARCHAR2 DEFAULT 'DEFAULT')

RETURN VARCHAR2;
```

restab

Optionally, specify the name of the expansion table to store the results. This table must be of type EXP TAB which the system defines as follows:

```
type exp_rec is record (
    xrel varchar2(12),
    xlevel number,
    xphrase varchar2(256)
);
type exp_tab is table of exp_rec index by binary_integer;
```

See Also:

"CTX_THES Result Tables and Data Types" in Oracle Text Result Tables for more information about EXP TAB.

phrase

Specify a phrase to lookup in thesaurus.

lvl

Specify how many levels of narrower terms to return. For example 2 means get the narrower terms of the narrower terms of the phrase.

tname

Specify the thesaurus name. If not specified, system default thesaurus is used.

Returns

This function returns a string of narrower terms instance in the form:

```
{nt1}|{nt2}|{nt3} ...
```

Example

To look up the narrower terms instance for *cat* down to two levels, enter the following statements:

```
declare
  terms varchar2(2000);
begin
  terms := ctx_thes.nti('CAT', 2, 'MY_THES');
  dbms_output_put_line('the narrower expansion for CAT is: '||terms);
end;
```

Related Topics

```
"OUTPUT STYLE"
```

"Narrower Term (NT_ NTG_ NTP_ NTI)"

14.21 NTP

This function returns all narrower terms partitive of a phrase as recorded in the specified thesaurus.

Syntax 1: Table Result

Syntax 2: String Result

restab

Optionally, specify the name of the expansion table to store the results. This table must be of type EXP TAB which the system defines as follows:

```
type exp_rec is record (
    xrel varchar2(12),
    xlevel number,
    xphrase varchar2(256)
);
type exp tab is table of exp rec index by binary integer;
```

See Also:

"CTX_THES Result Tables and Data Types" in Oracle Text Result Tables for more information about ${\tt EXP}\ {\tt TAB}.$

phrase

Specify a phrase to lookup in thesaurus.

lvl

Specify how many levels of narrower terms to return. For example 2 means get the narrower terms of the narrower terms of the phrase.

tname

Specify the thesaurus name. If not specified, system default thesaurus is used.

Returns

This function returns a string of narrower terms partitive in the form:

```
{nt1}|{nt2}|{nt3} ...
```

Example

To look up the narrower terms partitive for *cat* down to two levels, enter the following statements:



```
declare
  terms varchar2(2000);
begin
  terms := ctx_thes.ntp('CAT', 2, 'MY_THES');
  dbms_output.put_line('the narrower expansion for CAT is: '||terms);
end;
```

Related Topics

```
"OUTPUT_STYLE"
```

"Narrower Term (NT_ NTG_ NTP_ NTI)"

14.22 OUTPUT STYLE

Sets the output style for the return string of the CTX_THES expansion functions. This procedure has no effect on the table results to the CTX_THES expansion functions.

Syntax

showlevel

Specify TRUE to show level in BT/NT expansions.

showqualify

Specify TRUE to show phrase qualifiers.

showpt

Specify TRUE to show preferred terms with an asterisk *.

showid

Specify TRUE to show phrase ids.

Notes

The general syntax of the return string for CTX THES expansion functions is:

```
{pt indicator:phrase (qualifier):level:phraseid}
```

Preferred term indicator is an asterisk then a colon at the start of the phrase. The qualifier is in parentheses after a space at the end of the phrase. Level is a number.

The following is an example return string for turkey the bird:

```
*:TURKEY (BIRD):1:1234
```

14.23 PT

This function returns the preferred term of a phrase as recorded in the specified thesaurus.

Syntax 1: Table Result

Syntax 2: String Result

```
CTX_THES.PT(phrase IN VARCHAR2,
tname IN VARCHAR2 DEFAULT 'DEFAULT')
RETURN varchar2;
```

restab

Optionally, specify the name of the expansion table to store the results. This table must be of type EXP_TAB which the system defines as follows:

```
type exp_rec is record (
    xrel varchar2(12),
    xlevel number,
    xphrase varchar2(256)
);
type exp tab is table of exp rec index by binary integer;
```

See Also:

"CTX_THES Result Tables and Data Types" in Oracle Text Result Tables for more information about EXP_TAB.

phrase

Specify a phrase to lookup in thesaurus.

tname

Specify thesaurus name. If not specified, system default thesaurus is used.

Returns

This function returns the preferred term as a string in the form:

{pt}

Example

Consider a thesaurus MY_THES with the following preferred term definition for automobile:

```
AUTOMOBILE
PT CAR
```

To look up the preferred term for *automobile*, execute the following code:

```
declare
  terms varchar2(2000);
begin
  terms := ctx_thes.pt('AUTOMOBILE','MY_THES');
  dbms_output.put_line('The preferred term for automobile is: '||terms);
end;
```



Related Topics

```
"OUTPUT STYLE"
```

"Preferred Term (PT)"

14.24 RT

This function returns the related terms of a term in the specified thesaurus.

Syntax 1: Table Result

Syntax 2: String Result

restab

Optionally, specify the name of the expansion table to store the results. This table must be of type EXP TAB which the system defines as follows:

```
type exp_rec is record (
    xrel varchar2(12),
    xlevel number,
    xphrase varchar2(256)
);
type exp_tab is table of exp_rec index by binary_integer;
```

See Also:

"CTX_THES Result Tables and Data Types" in Oracle Text Result Tables for more information about EXP TAB.

phrase

Specify a phrase to lookup in thesaurus.

tname

Specify the thesaurus name. If not specified, system default thesaurus is used.

Returns

This function returns a string of related terms in the form:

```
{rt1}|{rt2}|{rt3}| ...
```

Example

Consider a thesaurus $\texttt{MY_THES}$ with the following related term definition for dog:

```
DOG
RT WOLF
RT HYENA
```



To look up the related terms for dog, execute the following code:

```
declare
  terms varchar2(2000);
begin
  terms := ctx_thes.rt('DOG','MY_THES');
  dbms_output.put_line('The related terms for dog are: '||terms);
end;
```

This codes produces the following output:

```
The related terms for dog are: {dog}|{wolf}|{hyena}
```

Related Topics

```
"OUTPUT_STYLE"
```

"Related Term (RT)"

14.25 SN

This function returns the scope note of the given phrase.

Syntax

```
CTX_THES.SN(phrase IN VARCHAR2,
tname IN VARCHAR2 DEFAULT 'DEFAULT')
RETURN VARCHAR2;
```

phrase

Specify a phrase to lookup in thesaurus.

tname

Specify the thesaurus name. If not specified, system default thesaurus is used.

Returns

This function returns the scope note as a string.

Example

```
declare
  note varchar2(80);
begin
  note := ctx_thes.sn('camera','mythes');
  dbms_output.put_line('CAMERA');
  dbms_output.put_line('SN'||note);
end;
sample output:

CAMERA
SN Optical cameras
```

14.26 SYN

This function returns all synonyms of a phrase as recorded in the specified thesaurus.

Syntax 1: Table Result

Syntax 2: String Result

```
CTX_THES.SYN(phrase IN VARCHAR2,
tname IN VARCHAR2 DEFAULT 'DEFAULT')
RETURN VARCHAR2;
```

restab

Optionally, specify the name of the expansion table to store the results. This table must be of type EXP TAB which the system defines as follows:

```
type exp_rec is record (
    xrel varchar2(12),
    xlevel number,
    xphrase varchar2(256)
);
type exp_tab is table of exp_rec index by binary_integer;
```

See Also:

"CTX_THES Result Tables and Data Types" in Oracle Text Result Tables for more information about $\texttt{EXP}_\texttt{TAB}$.

phrase

Specify a phrase to lookup in thesaurus.

tname

Specify the thesaurus name. If not specified, system default thesaurus is used.

Returns

This function returns a string of the form:

```
\{syn1\} | \{syn2\} | \{syn3\} \dots
```

Example

String Result

Consider a thesaurus named ANIMALS that has an entry for cat as follows:

```
CAT
SYN KITTY
SYN FELINE
```

To look-up the synonym for *cat* and obtain the result as a string, enter the following statements:

```
declare
  synonyms varchar2(2000);
begin
  synonyms := ctx_thes.syn('CAT','ANIMALS');
  dbms_output.put_line('the synonym expansion for CAT is: '||synonyms);
end:
```



This code produces the following output:

```
the synonym expansion for CAT is: {CAT}|{KITTY}|{FELINE}
```

Table Result

The following code looks up the synonyms for *canine* and obtains the results in a table. The contents of the table are printed to the standard output.

```
declare
  xtab ctx_thes.exp_tab;
begin
  ctx_thes.syn(xtab, 'canine', 'my_thesaurus');
  for i in 1..xtab.count loop
    dbms_output.put_line(lpad(' ', 2*xtab(i).xlevel) ||
    xtab(i).xrel || ' ' || xtab(i).xphrase);
  end loop;
end;
```

This code produces the following output:

```
PHRASE CANINE
PT DOG
SYN PUPPY
SYN MUTT
SYN MONGREL
```

Related Topics

```
"OUTPUT_STYLE"

"SYNonym (SYN)"
```

14.27 THES TT

This procedure finds and returns all top terms of a thesaurus. A top term is defined as any term which has a narrower term but has no broader terms.

This procedure differs from TT in that TT takes in a phrase and finds the top term for that phrase, but THES TT searches the whole thesaurus and finds all top terms.

Large Thesauri

Because this procedure searches the whole thesaurus, it can take some time on large thesauri. Oracle recommends that you not call this often for such thesauri. Instead, your application should call this once, store the results in a separate table, and use those stored results.

Syntax

restab

Specify the name of the expansion table to store the results. This table must be of type ${\tt EXP}$ TAB which the system defines as follows:



```
type exp_rec is record (
    xrel varchar2(12),
    xlevel number,
    xphrase varchar2(256)
);
type exp_tab is table of exp_rec index by binary_integer;
```



"CTX_THES Result Tables and Data Types" in Oracle Text Result Tables for more information about EXP TAB.

tname

Specify the thesaurus name. If not specified, system default thesaurus is used.

Returns

This procedure returns all top terms and stores them in restab.

14.28 TR

For a given mono-lingual thesaurus, this function returns the foreign language equivalent of a phrase as recorded in the thesaurus.



Foreign language translation is not part of the ISO-2788 or ANSI Z39.19 thesaural standards. The behavior of TR is specific to Oracle Text.

Syntax 1: Table Result

Syntax 2: String Result

```
CTX_THES.TR(phrase IN VARCHAR2,

lang IN VARCHAR2 DEFAULT NULL,

tname IN VARCHAR2 DEFAULT 'DEFAULT')

RETURN VARCHAR2;
```

restab

Optionally, specify the name of the expansion table to store the results. This table must be of type EXP TAB which the system defines as follows:

```
type exp_rec is record (
    xrel varchar2(12),
    xlevel number,
    xphrase varchar2(256)
);
type exp_tab is table of exp_rec index by binary_integer;
```



See Also:

"CTX_THES Result Tables and Data Types" in Oracle Text Result Tables for more information about EXP_TAB.

phrase

Specify a phrase to lookup in thesaurus.

lang

Specify the foreign language. Specify 'ALL' for all translations of phrase.

tname

Specify the thesaurus name. If not specified, system default thesaurus is used.

Returns

This function returns a string of foreign terms in the form:

```
{ft1}|{ft2}|{ft3} ...
```

Example

Consider a thesaurus MY THES with the following entries for cat:

```
SPANISH: gato
FRENCH: chat
SYN lion
SPANISH: leon
```

To look up the translation for *cat*, enter the following statements:

```
declare
  trans    varchar2(2000);
  span_trans varchar2(2000);
begin
  trans := ctx_thes.tr('CAT','ALL','MY_THES');
  span_trans := ctx_thes.tr('CAT','SPANISH','MY_THES')
  dbms_output.put_line('the translations for CAT are: '||trans);
  dbms_output.put_line('the Spanish translations for CAT are: '||span_trans);
end;
```

This codes produces the following output:

```
the translations for CAT are: {CAT}|{CHAT}|{GATO} the Spanish translations for CAT are: {CAT}|{GATO}
```

Related Topics

```
"OUTPUT_STYLE"
```

"Translation Term (TR)"

14.29 TRSYN

For a given mono-lingual thesaurus, this function returns the foreign equivalent of a phrase, synonyms of the phrase, and foreign equivalent of the synonyms as recorded in the specified thesaurus.



Foreign language translation is not part of the ISO-2788 or ANSI Z39.19 thesaural standards. The behavior of TRSYN is specific to Oracle Text.

Syntax 1: Table Result

Syntax 2: String Result

```
CTX_THES.TRSYN(phrase IN VARCHAR2,

lang IN VARCHAR2 DEFAULT NULL,

tname IN VARCHAR2 DEFAULT 'DEFAULT')
RETURN VARCHAR2;
```

restab

Optionally, specify the name of the expansion table to store the results. This table must be of type EXP TAB which the system defines as follows:

```
type exp_rec is record (
    xrel varchar2(12),
    xlevel number,
    xphrase varchar2(256)
);
type exp tab is table of exp rec index by binary integer;
```

See Also:

"CTX_THES Result Tables and Data Types" in Oracle Text Result Tables for more information about $\texttt{EXP}_\texttt{TAB}$.

phrase

Specify a phrase to lookup in thesaurus.

lang

Specify the foreign language. Specify 'ALL' for all translations of *phrase*.

tname

Specify the thesaurus name. If not specified, system default thesaurus is used.

Returns

This function returns a string of foreign terms in the form:



```
{ft1}|{ft2}|{ft3} ...
```

Example

Consider a thesaurus MY THES with the following entries for cat:

```
SPANISH: gato
FRENCH: chat
SYN lion
SPANISH: leon
```

To look up the translation and synonyms for *cat*, enter the following statements:

```
declare
   synonyms   varchar2(2000);
   span_syn   varchar2(2000);
begin
   synonyms := ctx_thes.trsyn('CAT','ALL','MY_THES');
   span_syn := ctx_thes.trsyn('CAT','SPANISH','MY_THES')
   dbms_output.put_line('all synonyms for CAT are: '||synonyms);
   dbms_output.put_line('the Spanish synonyms for CAT are: '||span_syn);
end;
```

This codes produces the following output:

```
all synonyms for CAT are: {CAT}|{CHAT}|{GATO}|{LION}|{LEON} the Spanish synonyms for CAT are: {CAT}|{GATO}|{LION}|{LEON}
```

Related Topics

```
"OUTPUT STYLE"
```

"Translation Term Synonym (TRSYN)"

14.30 TT

This function returns the top term of a phrase as recorded in the specified thesaurus.

Syntax 1: Table Result

Syntax 2: String Result

restab

Optionally, specify the name of the expansion table to store the results. This table must be of type EXP TAB which the system defines as follows:

```
type exp_rec is record (
    xrel varchar2(12),
    xlevel number,
    xphrase varchar2(256)
```



```
);
type exp_tab is table of exp_rec index by binary_integer;
```



"CTX_THES Result Tables and Data Types" in Oracle Text Result Tables for more information about EXP TAB.

phrase

Specify a phrase to lookup in thesaurus.

tname

Specify the thesaurus name. If not specified, system default thesaurus is used.

Returns

This function returns the top term string in the form:

{tt}

Example

Consider a thesaurus MY_THES with the following broader term entries for dog:

```
DOG
BT1 CANINE
BT2 MAMMAL
BT3 VERTEBRATE
BT4 ANIMAL
```

To look up the top term for *DOG*, execute the following code:

```
declare
  terms varchar2(2000);
begin
  terms := ctx_thes.tt('DOG','MY_THES');
  dbms_output.put_line('The top term for DOG is: '||terms);
end;
```

This code produces the following output:

```
The top term for dog is: {ANIMAL}
```

Related Topics

```
"OUTPUT_STYLE "
"Top Term (TT)"
```

14.31 UPDATE_TRANSLATION

Use this procedure to update an existing translation.

Syntax

```
CTX_THES.UPDATE_TRANSLATION(tname in varchar2, phrase in varchar2, language in varchar2,
```



```
translation in varchar2,
new translation in varchar2);
```

tname

Specify the name of the thesaurus, using no more than 30 characters.

phrase

Specify the phrase in the thesaurus to which to update a translation. The phrase must already exist in the thesaurus or an error is raised.

language

Specify the language of the translation, using no more than 10 characters.

translation

Specify the translated term to update. If no such translation exists, an error is raised. You can specify NULL if there is only one translation for the *phrase*. An error is raised if there is more than one translation for the term in the specified language.

new translation

Optionally, specify the new form of the translated term.

Example

The following code updates the Spanish translation for dog:

```
begin
   ctx_thes.update_translation('my_thes', 'dog', 'SPANISH:', 'PERRO', 'CAN');
end;
```



CTX_ULEXER Package

This chapter provides reference information on how to use the CTX_ULEXER PL/SQL package with the user-defined lexer.

CTX ULEXER declares the following type:

Name	Description
WILDCARD_TAB	Index-by table type that you use to specify the offset of characters to be treated as wildcard characters by the user-defined lexer query procedure.



The APIs in the $\mathtt{CTX_ULEXER}$ package do not support identifiers that are prefixed with the schema or the owner name.

15.1 WILDCARD_TAB

TYPE WILDCARD TAB IS TABLE OF NUMBER INDEX BY BINARY INTEGER;

Use this index-by table type to specify the offset of those characters in the query word to be treated as wildcard characters by the user-defined lexer query procedure.

Character offset information follows USC-2 codepoint semantics.

DBMS_SEARCH Package

The $\DBMS_SEARCH\ PL/SQL\ package\ provides\ procedures\ and\ functions\ to\ create,\ manage,\ and\ query\ a\ ubiquitous\ search\ index.$

Instead of creating various individual indexes or manually defining the <code>USER_DATASTORE</code> or <code>MULTI_COLUMN_DATASTORE</code> procedures along with materialized views, you can create a ubiquitous search index to automatically synthesize multiple tables or views into a single index. This index lets you perform full-text and range-based searches across multiple objects within an entire schema.



In addition to the examples provided for each of these APIs, you can run various end-to-end example scenarios, as demonstrated in *Oracle Text Application Developer's Guide*.

Name	Description
CREATE_INDEX	Creates a ubiquitous search index.
ADD_SOURCE	Adds a table, view, or Duality view to the index as data source.
REMOVE_SOURCE	Removes a data source and all its associated data from the index.
DROP_INDEX	Removes the index and all its associated data from the database.
GET_DOCUMENT	Returns a virtual indexed JSON document for the specified source metadata.
FIND	Retrieves a hitlist, and facets an aggregations of JSON documents based on the specified filter conditions.

16.1 CREATE INDEX

The DBMS_SEARCH.CREATE_INDEX procedure creates a ubiquitous search index (or DBMS_SEARCH index) to perform full-text and range-based searches across multiple schema objects.

The index type is a JSON search index enabled with a predefined set of preferences and settings to perform full text searches on tables, views, and Duality views.

Syntax

```
DBMS_SEARCH.CREATE_INDEX(
   index_name IN VARCHAR2,
   tablespace IN VARCHAR2 DEFAULT,
   datatype IN VARCHAR2 DEFAULT
);
```

index_name

Specify the name of the DBMS_SEARCH index to create. You can also specify the schema owner name along with the index name as:

[schema].index name

tablespace

Specify the name of the tablespace to contain the index or index partitions.

datatype

Specify the datatype of the DATA column on which to create the index. The allowed values are $_{\rm JSON}$ and $_{\rm OSON}$.

The default value is set to JSON, so you need to specify the datatype argument only if you want to override this default.

Notes

- You can define which tables or views should be indexed by adding them as data sources
 into your index. All the columns of those tables or views are indexed. Use the

 DBMS_SEARCH.ADD_SOURCE and DBMS_SEARCH.REMOVE_SOURCE procedures to manage data
 sources.
- The DBMS SEARCH index is created with the following default indexing preferences:

Preference	Description
WILDCARD_INDEX	Enables wildcard indexing for a fast wildcard search.
BASIC_WORDLIST	Enables stemming and fuzzy matching.
SEARCH_ON	Allows both full-text and range-search queries for a specific data type. The supported data types are NUMBER (for indexing numeric values) and TIMESTAMP (for indexing date-time values).
SYNC and OPTIMIZE	Creates background jobs at predefined intervals to automatically synchronize the DML changes and optimize the index using the AUTO_DAILY mode on all data sources.
	You do not need to explicitly run any SYNC_INDEX and OPTIMIZE_INDEX operations on this index.

- You can query this index using the CONTAINS(), JSON_TEXTCONTAINS(), and JSON_EXISTS
 operators on the INDEX NAME table.
- You can use the following DBMS SEARCH dictionary views to examine these indexes:
 - USER_DBMS_SEARCH_INDEXES: To query information about the DBMS_SEARCH indexes that
 are created in a user's schema.
 - ALL_DBMS_SEARCH_INDEXES: To query information about all existing DBMS_SEARCH indexes, corresponding to each index owner.
 - USER_DBMS_SEARCH_INDEX_SOURCES: To query information about the data sources that are added to the DBMS_SEARCH indexes, created in a user's schema.
 - ALL_DBMS_SEARCH_INDEX_SOURCES: To query information about all existing data sources added to the DBMS_SEARCH indexes, corresponding to each index owner.

Example

This example specifies the <code>index_name</code>, <code>tablespace</code>, and <code>datatype</code> arguments. Here, the schema owner name is specified along with the index name as <code>SCOTT.MYINDEX</code>.

```
CREATE TABLESPACE tbs_02 DATAFILE 'dt.dbf' size 100MB segment space management auto; exec DBMS SEARCH.CREATE INDEX('SCOTT.MYINDEX','tbs 02','JSON');
```



Related Topics

- Oracle Text Application Developer's Guide
- Oracle Database SQL Language Reference
- Oracle Text Views
 This is a list of all the views provided by Oracle Text.

16.2 ADD SOURCE

The DBMS_SEARCH.ADD_SOURCE procedure adds one or more data sources (tables or views) from different schemas to the DBMS_SEARCH index.

Syntax

```
DBMS_SEARCH.ADD_SOURCE(
  index_name     IN      VARCHAR2,
  source_name     IN      VARCHAR2
);
```

index name

Specify name of the index to which you want to add the table or view. You can also specify $[schema].index_name$.

source name

Specify name of the table, view, or Duality view to add to the index. You can also specify [schema].table_or_view_name.

Notes

- To add a data source, the index owner must have SELECT and DML access to the source. The user also must have the CTXAPP privilege.
- All the data sources (such as table, view, or each table in the view definition) that are
 added to the DBMS_SEARCH index must include at least one Primary Key column. Each table
 that is part of a view source having a foreign key must also have the Foreign Key
 constraint, referencing the relevant primary keys defined on the table.
 - If the source table does not have a primary key, then a ROWID is used instead. However, Oracle strongly recommends defining a primary key.
- The DBMS_SEARCH index stores all supported SQL data types (including Object Type columns) in JSON objects, except for the XMLTYPE and LONG data types. This means that you cannot add a table or view as a data source to the index if it has a column with the XMLTYPE or LONG data type. The maximum allowed length of a JSON data type is 32 megabytes.
- You can use the USER_DBMS_SEARCH_INDEX_SOURCES and ALL_DBMS_SEARCH_INDEX_SOURCES
 dictionary views to query information about the data sources that are added to your
 DBMS_SEARCH_indexes.



Examples

```
exec DBMS_SEARCH.ADD_SOURCE('MYINDEX','MYTABLE');

exec DBMS_SEARCH.ADD_SOURCE('MYINDEX','MYVIEW');

exec DBMS_SEARCH.ADD_SOURCE('DOCUSER.MYINDEX','DOCUSER.MYTABLE');
```

Related Topics

- Oracle Text Application Developer's Guide
- Oracle Text Views
 This is a list of all the views provided by Oracle Text.

16.3 REMOVE SOURCE

The DBMS_SEARCH.REMOVE_SOURCE procedure removes one or more data sources (tables or views) from the DBMS_SEARCH index.

When run, this procedure deletes all indexed data and stops further indexing or maintenance operations on the associated data sources (tables or views).

Syntax

```
DBMS_SEARCH.REMOVE_SOURCE(
  index_name VARCHAR2,
  source name VARCHAR2);
```

index name

Specify name of the index from which you want to remove the table or view.

source_name

Specify name of the table or view that you want to remove.

Example

```
exec DBMS SEARCH.REMOVE SOURCE('MYINDEX', 'MYTABLE');
```

Related Topics

Oracle Text Application Developer's Guide

16.4 DROP_INDEX

The DBMS_SEARCH.DROP_INDEX procedure removes a DBMS_SEARCH index and all its associated data from the database.

Syntax

```
DBMS_SEARCH.DROP_INDEX(
    INDEX_NAME VARCHAR2);
```

index name

Specify name of the index that you want to drop.

Example

```
exec DBMS SEARCH.DROP INDEX('MYINDEX');
```

Related Topics

Oracle Text Application Developer's Guide

16.5 GET DOCUMENT

The DBMS_SEARCH.GET_DOCUMENT procedure returns a virtual indexed JSON document as is indexed in the JSON search index for a particular row of an indexed data source (table or view).

The DBMS_SEARCH index references associated data source tables to dynamically create a **virtual indexed document**. This document contains a JSON representation for each indexed row of a table or view that is added as a data source to this index. In this way, you can view all the contents extracted from the original base tables.

Syntax

```
DBMS_SEARCH.GET_DOCUMENT(
  index_name VARCHAR2,
  metadata JSON
);
```

index name

Specify name of the index for which you want to retrieve the data.

metadata

Specify the JSON metadata values, such as <code>owner</code>, <code>source</code>, or <code>key</code>. You must specify the metadata format based on the <code>metadata</code> column of the <code>index</code> <code>name</code> table.

Example

```
SELECT DBMS SEARCH.GET DOCUMENT('MYINDEX', METADATA) from MYINDEX;
```



Related Topics

Oracle Text Application Developer's Guide

16.6 FIND

The DBMS_SEARCH.FIND procedure retrieves a hitlist, and facets an aggregations of JSON documents based on the specified query-by-example (QBE) filter conditions.

You can compute aggregations on different fields of the JSON data. The query lists search results in the JSON Results Set Interface, which supports faceted navigation and aggregations.

Syntax

```
DBMS_SEARCH.FIND(
  index_name VARCHAR2,
  search_QBE JSON);
```

index name

Specify name of the index on which you want to perform the query.

search_QBE

Specify the result_set_descriptor parameter value in JSON. It describes what the result set should contain.

The JSON format input result set descriptor consists of the \$query, \$search, and \$facet parts:

```
{
   "$query":text query and filter conditions,
   "$search":search result specifications,
   "$facet":faceted result specifications
}
```

For details on each of these JSON objects, see The JSON Format Input Result Set Descriptor.

The JSON format output result set descriptor consists of the following parts:

```
"$count":number
"$hit":[ hit_object_1, ..., hit_object_i , ... ]
"$facet":[ facet_object_1, ..., facet_object_i, ...]
```

For details on each of these JSON objects, see The JSON Format Result Set Output.

Example

Create a table and populate it with values:

```
SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 80
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 100
```



```
connect sys/knl example as sysdba;
Connected.
grant connect, resource, unlimited tablespace, ctxapp
  to u1 identified by u1;
Grant succeeded.
connect u1/u1;
Connected.
create table tbl(id number primary key, jsn col clob check(jsn col is json));
Table created.
INSERT INTO tbl
 VALUES (1, '{ "zebra" : { "price" : [2000,1000],
                          "name" : "Marty",
                          "stripes" : ["Dark", "Light"],
                          "handler" : "Bob", "sold" : true }}');
1 row created.
INSERT INTO tbl
  VALUES (2, '{ "zebra" : { "rating": 5, "price" : 1000,
                          "name" : "Zigby",
                          "stripes" : ["Light", "Grey"],
                          "handler" : "Handy Marty", "sold" : "true" }}');
1 row created.
 INSERT INTO tbl
  VALUES (3, '{ "zebra" : { "rating": 4.5, "price" : 3000,
                           "name" : "Zigs",
                           "stripes" : ["Grey", "Dark"],
                           "handler" : "Handy Marty", "sold" : false }}');
1 row created.
 INSERT INTO tbl
  VALUES (4,'{ "zebra" : { "rating": "4.5", "price" : "3000",
                           "name" : "Zigs",
                           "stripes" : ["Grey", "Dark"],
                           "handler" : "Handy Marty", "sold" : null }}');
1 row created.
 commit;
Commit complete.
Create a DBMS SEARCH index using the DBMS SEARCH. CREATE INDEX procedure, and add a
source table to the index:
SQL> exec DBMS SEARCH.CREATE INDEX('JIDX');
PL/SQL procedure successfully completed.
SQL> exec DBMS SEARCH.ADD SOURCE('JIDX', 'TBL');
PL/SQL procedure successfully completed.
SQL>
```

Run the DBMS SEARCH.FIND procedure:

```
Query: All zebras having name starting with Zig or having name Marty and
having a price greater than equal to 2000
Facets: For all zebras that satisfy the query, do the following
-- 1. Get the count of zebras per zebra handler
-- 2. Get the minimum zebra rating
-- 3. Get the count of zebras for each unique stripe color
select DBMS SEARCH.FIND('JIDX', JSON('
    "$query": { "$and" : [
                   { "U1.TBL.JSN COL.zebra.name" : { "$contains" : "Zig% or
Marty" } },
                   { "U1.TBL.JSN COL.zebra.price" : { "$gte" : 2000 } }
                },
      "$facet" : [
        { "$uniqueCount" : "U1.TBL.JSN COL.zebra.handler" },
        { "$min" : "U1.TBL.JSN COL.zebra.rating" },
        { "$uniqueCount" : "U1.TBL.JSN COL.zebra.stripes" }
    }'));
```

The output is as follows:

```
FIND RESULT
 "$count" : 3,
 "$facet" :
      "U1.TBL.JSN COL.zebra.handler" :
          "value" : "Handy Marty",
          "$uniqueCount" : 2
        },
          "value" : "Bob",
          "$uniqueCount" : 1
     1
    },
      "U1.TBL.JSN COL.zebra.rating" :
        "$min" : 4.5
    },
      "U1.TBL.JSN COL.zebra.stripes" :
```

```
{
    "value": "Dark",
    "$uniqueCount": 3
},
{
    "value": "Grey",
    "$uniqueCount": 2
},
{
    "value": "Light",
    "$uniqueCount": 1
}
]
}
]
}
1 row selected.
connect sys/knl_example as sysdba;
Connected.
drop user u1 cascade;
User dropped.
```

Related Topics

RESULT SET

This procedure executes an XML or JSON query and generates a result set in XML or JSON.

Oracle Database Introduction to Simple Oracle Document Access (SODA)



Oracle Text Utilities

Oracle Text provides utilities for managing and operating on Text indexes. For example, you can load a specific thesaurus into the index, and you can create your own knowledge base to be associated with the index, among other things. This chapter discusses the utilities shipped with Oracle Text.

The following topics are included:

- Thesaurus Loader (ctxload)
- Entity Extraction User Dictionary Loader (ctxload)
- Knowledge Base Extension Compiler (ctxkbtc)
- Lexical Compiler (ctxlc)



The APIs in the utilities shipped with Oracle Text do not support identifiers that are prefixed with the schema or the owner name.

17.1 Thesaurus Loader (ctxload)

Use ctxload to import a thesaurus file into the Oracle Text thesaurus tables.

An import file is an ASCII flat file that contains entries for synonyms, broader terms, narrower terms, or related terms, which can be used to expand queries.

This section contains the following topics.

- ctxload Text Loading
- ctxload Syntax
- ctxload Examples



For examples of import files for thesaurus importing, see "Structure of ctxload Thesaurus Import File" in Text Loading Examples for Oracle Text

17.1.1 ctxload Text Loading

The ctxload program no longer supports the loading of text columns. To load files to a text column in batch mode, Oracle recommends that you use SQL*Loader.

"SQL*Loader Example" in Text Loading Examples for Oracle Text

17.1.2 ctxload Syntax

ctxload Mandatory Arguments

-user

Specify the user name and password of the user running ctxload.

The user name and password can be followed immediately by $@sqlnet_address$ to permit logging on to remote databases. The value for $sqlnet_address$ is a database connect string. If the to_TASK environment variable is set to a remote database, then you do not need to specify a value for $sqlnet_address$ to connect to the database.

-name object_name

When you use ctxload to import a thesaurus, use object_name to specify the name of the thesaurus to be imported.

Use object name to identify the thesaurus in queries that use thesaurus operators.



Thesaurus name must be unique. If the name specified for the thesaurus is identical to an existing thesaurus, then <code>ctxload</code> returns an error and does not overwrite the existing thesaurus.

-file file name

When ctxload is used to import a thesaurus, use *file_name* to specify the name of the import file that contains the thesaurus entries.

When ctxload is used to export a thesaurus, use *file_name* to specify the name of the export file created by ctxload.



If the name specified for the thesaurus dump file is identical to an existing file, then ctxload overwrites the existing file.



ctxload Optional Arguments

-thes

Import a thesaurus. Specify the source file with the -file argument. Specify the name of the thesaurus to be imported with -name.

-thescase y | n

Specify y to create a case-sensitive thesaurus with the name specified by -name and populate the thesaurus with entries from the thesaurus import file specified by -file. If -thescase is y (the thesaurus is case-sensitive), ctxload enters the terms in the thesaurus exactly as they appear in the import file.

The default for -thescase is *n* (case-insensitive thesaurus).



-thescase is valid for use only with the -thes argument.

-thesdump

Export a thesaurus. Specify the name of the thesaurus to be exported with the -name argument. Specify the destination file with the -file argument.

-log

Specify the name of the log file to which ctxload writes any national-language supported (Globalization Support) messages generated during processing. If you do not specify a log file name, the messages appear on the standard output. The logs generated by ctxload will be present in $SORACLE\ HOME/ctx/log\ directory$.

-trace

Enables SQL statement tracing using ALTER SESSION SET SQL_TRACE TRUE. This command captures all processed SQL statements in a trace file, which can be used for debugging. The location of the trace file is operating-system dependent and can be modified using the DIAGNOSTIC DEST initialization parameter.



For more information about SQL trace and the <code>DIAGNOSTIC_DEST</code> initialization parameter, see $Oracle\ Database\ Administrator's\ Guide$

17.1.3 ctxload Examples

This section provides examples for some of the operations that ctxload can perform.



For more document loading examples, see Text Loading Examples for Oracle Text

The following example imports a thesaurus named tech_doc from an import file named tech thesaurus.txt:



```
ctxload -user jsmith/password -thes -name tech doc -file tech thesaurus.txt
```

The following example exports the contents of a thesaurus named tech_doc into a file named tech_thesaurus.out:

```
ctxload -user jsmith/password -thesdump -name tech_doc -file tech_thesaurus.out
```

17.2 Entity Extraction User Dictionary Loader (ctxload)

Use ctxload to import an entity extraction user dictionary into Oracle Text tables.

An import file is an XML flat file containing entries for entities, with their associated types and alternate forms.

This section contains the following topics.

- ctxload Syntax
- Considerations When Creating a User Dictionary
- XML Schema
- ctxload Example

17.2.1 ctxload Syntax

```
ctxload -user username[/password][@sqlnet_address]
    -extract
    -name entity extraction policy name
    -file user-dictionary file name
    [-drop] to drop a user-dictionary from a policy
```

ctxload Mandatory Arguments

-user

Specify the user name and password of the user running ctxload.

The user name and password can be followed immediately by @sqlnet_address to permit logging on to remote databases. The value for sqlnet_address is a database connect string. If the TWO_TASK environment variable is set to a remote database, then you do not need to specify a value for sqlnet address to connect to the database.

-name entity extraction policy name

When you use <code>ctxload</code> to import an entity extraction dictionary, use <code>object_name</code> to specify the entity extraction policy to associate the dictionary with. An entity extraction policy can have only one user dictionary.

-file user-dictionary file name

Use file to specify the name of the XML file containing the user dictionary.

-drop

Drop the user dictionary currently associated with an entity extraction policy.

17.2.2 Considerations When Creating a User Dictionary

The following are some considerations for when creating a user dictionary:

Entity mentions are case-sensitive. They cannot contain any null characters.

- Entity type names are case-insensitive. They cannot contain any null or comma characters.
- Customers will be able to assign two or more entity types to a single entity mention. For example, the entity "Washington" could be assigned the type "CITY" and also the type "STATE".
- The content of a user's dictionary is invisible to other users.
- The maximum byte length of an entity mention is 512 bytes by the server-side database character set.
- The maximum byte length of an entity type name is 30 bytes by the server-side database character set.

17.2.3 XML Schema

The entity extraction dictionary follows this XML schema:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="dictionary">
 <xsd:complexType>
   <xsd:sequence>
     <xsd:element name="entities" type="entityType" maxOccurs="unbounded"/>
   </xsd:sequence>
  <xsd:complexType>
</xsd:element>
<xsd:complexType name="entityType">
  <xsd:sequence>
    <xsd:element name="entity" type="entType" maxOccurs="unbounded"/>
 </xsd:sequence>
  </xsd:attribute name="language" type="xsd:string"/>
</xsd:complexType>
<xsd:complexType name="entType">
  <xsd:sequence>
    <xsd:element name="value" type="xsd:string"/>
   <xsd:element name="type" type="xsd:string" minOccurs="1" maxOccurs="unbounded"/>
   <xsd:element name="alternate" type="xsd:string" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
 </xsd:complexType>
</xsd:schema>
```

The following tables illustrate some aspects of the XML schema for the entity extraction dictionary.

Element Name	Description	
dictionary	Collection of entities	
entities	Collection of entities per language	
entity	Each entity	
value	Entity mention	
type	Entity type	
alternate	Alternate form of entity	



Attribute Name	Element Name	Description
language	Entities	Language name of each entity in entities

17.2.4 ctxload Example

The following is an example of an entity extraction user dictionary file that can be loaded using

```
<?xml version="1.0" encoding="utf-8"?>
<dictionary>
 <entities>
    <entity>
     <value>New York</value>
     <type>city</type>
   </entity>
 </entities>
  <entities language="german">
   <entity>
     <value>Deutschland</value>
     <type>country</type>
   <entity>
 </entities>
  <entities language="english">
   <entity>
     <value>Astra</value>
       <type>person</type>
        <type>organization</type>
    </entity>
    <entity>
      <value>George W. Bush<value>
        <type>person</type>
      <alternate>G. W. Bush<alternate>
      <alternate>G. Bush<alternate>
    </entity>
  </entities>
</dictionary>
```

17.3 Knowledge Base Extension Compiler (ctxkbtc)

The knowledge base is the information source that Oracle Text uses to perform theme analysis, such as theme indexing, processing ABOUT queries, and to document theme extraction with the $\texttt{CTX}_\texttt{DOC}$ package. A knowledge base is supplied for English and French and is installed by default.

With the ctxkbtc compiler, you can:

- Extend your knowledge base by compiling one or more thesauri with the Oracle Text
 knowledge base. The extended information can be application-specific terms and
 relationships. During theme analysis, the extended portion of the knowledge base
 overrides any terms and relationships in the knowledge base where there is overlap.
- Create a new user-defined knowledge base by compiling one or more thesauri. In languages other than English and French, this feature can be used to create a languagespecific knowledge base.



Only CTXSYS can extend the knowledge base.

This section contains the following topics.

- Knowledge Base Character Set
- ctxkbtc Syntax
- ctxkbtc Usage Notes
- ctxkbtc Limitations
- ctxkbtc Constraints on Thesaurus Terms
- ctxkbtc Constraints on Thesaurus Relations
- Extending the Knowledge Base
 - Example for Extending the Knowledge Base
- Adding a Language-Specific Knowledge Base
 - Limitations for Adding a Knowledge Base
- Order of Precedence for Multiple Thesauri
- Size Limits for Extended Knowledge Base

See Also:

For more information about the ABOUT operator, see ABOUT operator in Oracle Text CONTAINS Query Operators

For more information about document services, see CTX_DOC Package

17.3.1 Knowledge Base Character Set

Knowledge bases can be in any single-byte character set. Supplied knowledge bases are in WE8ISO8859P1. You can store an extended knowledge base in another character set such as US7ASCII.

17.3.2 ctxkbtc Syntax

```
ctxkbtc -user uname/passwd
[-name thesname1 [thesname2 ... thesname16]]
[-revert]
[-stoplist stoplistname]
[-verbose]
[-log filename]
```

-user

Specify the user name and password for the administrator creating an extended knowledge base. This user must have write permission to the <code>ORACLE HOME</code> directory.

-name thesname1 [thesname2 ... thesname16]

Specify the names of the thesauri (up to 16) to be compiled with the knowledge base to create the extended knowledge base. The thesauri you specify must already be loaded with ctxload with the "-thescase Y" option.

-revert

Reverts the extended knowledge base to the default knowledge base provided by Oracle Text.

-stoplist stoplistname

Specify the name of the stoplist. Stopwords in the stoplist are added to the knowledge base as useless words that are prevented from becoming themes or contributing to themes. Add stopthemes after running this command using CTX DLL.ADD STOPTHEME.

-verbose

Displays all warnings and messages, including non-Globalization Support messages, to the standard output.

-log

Specify the log file for storing all messages. When you specify a log file, no messages are reported to standard out. The logs generated by ctxkbtc will be present in \$ORACLE HOME/ctx/log directory.

17.3.3 ctxkbtc Usage Notes

- Before running ctxkbtc, you must set the NLS_LANG environment variable to match the database character set.
- The user issuing ctxkbtc must have write permission to the ORACLE_HOME, because the
 program writes files to this directory.
- Before being compiled, each thesaurus must be loaded into Oracle Text case sensitive with the "-thescase Y" option in ctxload.
- Running ctxkbtc twice removes the previous extension.

17.3.4 ctxkbtc Limitations

The ctxkbtc program has the following limitations:

- When upgrading or downgrading your database to a different release, for theme indexing and related features to work correctly, Oracle recommends that you recompile your extended knowledge base in the new environment.
- Before extending the knowledge base, you must terminate all server processes that have invoked any knowledge base-related Text functions during their lifetime.
- There can be only one user extension for each language for each installation. Because a
 user extension affects all users at the installation, only the CTXSYS user can extend the
 knowledge base.
- In an Oracle RAC environment, the <code>ORACLE_HOME</code> can either be shared between multiple nodes, or each node can have its own <code>ORACLE_HOME</code>. The following requirements apply:
 - Before using any knowledge base-dependent functionality in any of the Oracle RAC nodes, ctxkbtc must be run in every ORACLE_HOME in the Oracle RAC environment.
 - When using ctxkbtc, the exact same input thesaurus content must be used in every ORACLE HOME in the Oracle RAC environment.



17.3.5 ctxkbtc Constraints on Thesaurus Terms

Terms are case sensitive. If a thesaurus has a term in uppercase, for example, the same term present in lowercase form in a document will not be recognized.

The maximum length of a term is 80 characters.

Disambiguated homographs are not supported.

17.3.6 ctxkbtc Constraints on Thesaurus Relations

The following constraints apply to thesaurus relations:

- BTG and BTP are the same as BT. NTG and NTP are the same as NT.
- Only preferred terms can have a BT, NTs or RTs.
- If a term has no USE relation, it will be treated as its own preferred term.
- If a set of terms are related by SYN relations, only one of them may be a preferred term.
- An existing category cannot be made a top term.
- There can be no cycles in BT and NT relations.
- A term can have at most one preferred term and at most one BT. A term may have any number of NTs.
- An RT of a term cannot be an ancestor or descendent of the term. A preferred term may have any number of RTs up to a maximum of 32.
- The maximum height of a tree is 16 including the top term level.
- When multiple thesauri are being compiled, a top term in one thesaurus should not have a broader term in another thesaurus.

Note:

The thesaurus compiler tolerates some violations of the preceding rules. For example, if a term has multiple BTs, then the compiler ignores all but the last one it encounters.

Similarly, BTs between existing knowledge base categories result only in a warning message.

Oracle recommends that you do not set up extended storage bases with violations. Using extended storage bases containing violations can produce undesired results.

17.3.7 Extending the Knowledge Base

Extend the supplied knowledge base by compiling one or more thesauri with the Oracle Text knowledge base. The extended information can be application-specific terms and relationships. During theme analysis, the extended portion of the knowledge base overrides any terms and relationships in the knowledge base where there is overlap.

When extending the knowledge base, Oracle recommends that new terms be linked to one of the categories in the knowledge base for best results in theme proving when appropriate. If new terms are kept completely disjoint from existing categories, fewer themes from new terms will be proven. The result of this is poorer precision and recall with ABOUT queries as well poor quality of gists and theme highlighting.

Link new terms to existing terms by making an existing term the broader term for the new terms.

17.3.8 Example for Extending the Knowledge Base

You purchase a medical thesaurus medthes containing a hierarchy of medical terms. The four top terms in the thesaurus are the following:

- Anesthesia and Analgesia
- Anti-Allergic and Respiratory System Agents
- · Anti-Inflammatory Agents, Antirheumatic Agents, and Inflammation Mediators
- Antineoplastic and Immunosuppressive Agents

To link these terms to the existing knowledge base, add the following entries to the medical thesaurus to map the new terms to the existing *health and medicine* branch:

```
health and medicine

NT Anesthesia and Analgesia

NT Anti-Allergic and Respiratory System Agents

NT Anti-Inflamammatory Agents, Antirheumatic Agents, and Inflamation Mediators

NT Antineoplastic and Immunosuppressive Agents
```

Set your globalization support language environment variable to match the database character set. For example, if your database character set is WE8ISO8859P1 and you are using American English, set your NLS LANG as follows:

```
setenv NLS LANG AMERICAN AMERICA.WE8ISO8859P1
```

Assuming the medical thesaurus is in a file called med.thes, load the thesaurus as medthes with ctxload as follows:

```
ctxload -thes -thescase y -name medthes -file med.thes -user ctxsys/ctxsys
```

To link the loaded thesaurus medthes to the knowledge base, use ctxkbtc as follows:

```
ctxkbtc -user ctxsys/ctxsys -name medthes
```

17.3.9 Adding a Language-Specific Knowledge Base

Extend theme functionality to languages other than English or French by loading your own knowledge base for any single-byte whitespace delimited language, including Spanish.

Theme functionality includes theme indexing, ABOUT queries, theme highlighting, and the generation of themes, gists, and theme summaries with the CTX DOC PL/SQL package.

Extend theme functionality by adding a user-defined knowledge base. For example, you can create a Spanish knowledge base from a Spanish thesaurus.

To load your language-specific knowledge base, follow these steps:

- 1. Load your custom thesaurus using ctxload.
- Set NLS_LANG so that the language portion is the target language. The charset portion must be a single-byte character set.

3. Compile the loaded thesaurus using ctxkbtc:

ctxkbtc -user ctxsys/ctxsys -name my_lang_thes

This command compiles your language-specific knowledge base from the loaded thesaurus. To use this knowledge base for theme analysis during indexing and ABOUT queries, specify the NLS LANG language as the THEME LANGUAGE attribute value for the BASIC LEXER preference.

17.3.10 Limitations for Adding a Knowledge Base

The following limitations hold for adding knowledge bases:

- Oracle Text supplies knowledge bases in English and French only. You must provide your own thesaurus for any other language.
- You can only add knowledge bases for languages with single-byte character sets. You
 cannot create a knowledge base for languages which can be expressed only in multibyte
 character sets. If the database is a multibyte universal character set, such as UTF-8, the
 NLS_LANG parameter must still be set to a compatible single-byte character set when
 compiling the thesaurus.
- Adding a knowledge base works best for whitespace delimited languages.
- You can have at most one knowledge base for each globalization support language.
- Obtaining hierarchical query feedback information such as broader terms, narrower terms and related terms does not work in languages other than English and French. In other languages, the knowledge bases are derived entirely from your thesauri. In such cases, Oracle recommends that you obtain hierarchical information directly from your thesauri.

17.3.11 Order of Precedence for Multiple Thesauri

When multiple thesauri are to be compiled, precedence is determined by the order in which thesauri are listed in the arguments to the compiler, assumed to be *most preferred* first. A user-defined thesaurus always has precedence over the built-in knowledge base.

17.3.12 Size Limits for Extended Knowledge Base

The following table lists the size limits associated with creating and compiling an extended knowledge base.

Table 17-1 Size Limit for the Extended Knowledge Base

Description of Parameter	Limit
Number of RTs (from + to) for each term	32
Number of terms for each single hierarchy (for example, all narrower terms for a given top term)	64000
Number of new terms in an extended knowledge base	1 million
Number of separate thesauri that can be compiled into a user extension to the KB	16

17.4 Lexical Compiler (ctxlc)

The Lexical Compiler (ctxlc) is a command-line utility that enables you to create your own Chinese and Japanese lexicons (dictionaries). Such a lexicon may either be generated from a

user-supplied word list or from the merging of a word list with the system lexicon for that language.

ctxlc creates the new lexicon in your current directory. The new lexicon consists of three files, drold.dat, drolk.dat, and droli.dat. To change your system lexicon for Japanese or Chinese, overwrite the system lexicon with these files.

The Lexical Compiler can also generate wordlists from the system lexicons for Japanese and Chinese, enabling you to see their contents. These word lists go to the standard output and thus can be redirected into a file of your choice.

After overwriting the system lexicon, you need to re-create your indexes before querying them.

This section contains the following topics.

- Syntax of ctxlc
- ctxlc Performance Considerations
- ctxlc Usage Notes
- ctxlc Example

17.4.1 Syntax of ctxlc

ctxlc has the following syntax:

```
ctxlc -ja | -zht [ -n ] -ics character_set -i input_file
ctxlc -ja | -zht -ocs character set [ > output file ]
```

ctxload Mandatory Arguments

-ja | -zht

Specify the language of the lexicon to modify or create. -ja indicates the Japanese lexicon; -zht indicates the Chinese lexicon, the same for either traditional or simplified Chinese.

-ics character set

Specify the character set of the input file denoted by -i input_file. input_file is the list of words, one word to a line, to use in creating the new lexicon.

-i input_file

Specify the file containing words to use in creating a new lexicon.

-ocs character set

Specify the character set of the text file to be output.

ctxload Optional Arguments

-n

Specify -n to create a new lexicon that consists only of user-supplied words taken from $input_file$. If -n is not specified, then the new lexicon consists of a merge of the system lexicon with $input_file$. Also, when -n is not selected, a text file called drolt.dat, is created in the current directory to enable you to inspect the contents of the merged lexicon without having to enter another ctxlc command.



17.4.2 ctxlc Performance Considerations

You can add up to 1,000,000 new words to a lexicon. However, creating a very large lexicon can reduce performance in indexing and querying. Performance is best when the lexicon character set is UTF-8. There is no performance impact on the Chinese or Japanese V-gram lexers, as they do not use lexicons.

17.4.3 ctxlc Usage Notes

Oracle recommends the following practices with regard to ${\tt ctxlc}$:

- Save your plain text dictionary file in your environment for emergency use.
- When upgrading or downgrading your database to a different release, recompile your plain text dictionary file in the new environment so that the user lexicon will work correctly.

17.4.4 ctxlc Example

In this example, you create a new Japanese lexicon from the file <code>jadict.txt</code>, a word list that uses the JA16EUC character set. Because you are not specifying <code>-n</code>, the new lexicon is the result of merging <code>jadict.txt</code> with the system Japanese lexicon. Then replace the existing Japanese lexicon with the new, merged one.

```
% ctxlc -ja -ics JA16EUC -i jadict.txt
```

This creates new files in the current directory:

```
% ls
drold.dat
drolk.dat
droli.dat
drolt.dat
```

The system lexicon files for Japanese and Chinese are named droldxx.dat droldxx.dat, and droldxx.dat, where xx is either JA (for Japanese) or ZH (for Chinese). Rename the three new files and copy them to the directory containing the system Japanese lexicon.

```
% mv drold.dat droldJA.dat
% mv drolk.dat drolkJA.dat
% mv droli.dat drolijA.dat
% cp *dat $ORACLE HOME/ctx/data/jalx
```

This replaces the system Japanese lexicon with one that is a merge of the old system lexicon and your wordlist from jadict.txt.

You can also use ctxlc to get a dump of a system lexicon. This example dumps the Chinese lexicon to a file called new chinese dict.txt in the current directory:

```
% ctxlc -zh -ocs UTF8 > new_chinese_dict.txt
```

This creates a file, new_japanese.dict.txt, using the UTF8 character set, in the current directory.



Oracle Text Alternative Spelling

This chapter describes various ways that Oracle Text handles alternative spelling of words. It also documents the alternative spelling conventions that Oracle Text uses in the German, Danish, and Swedish languages.

The following topics are covered:

- Overview of Alternative Spelling Features
- Overriding Alternative Spelling Features
- Alternative Spelling Conventions

18.1 Overview of Alternative Spelling Features

Some languages have alternative spelling forms for certain words. For example, the German word *Schoen* can also be spelled as *Schön*.

The form of a word is either *original* or *normalized*. The original form of the word is how it appears in the source document. The normalized form is how it is transformed, if it is transformed at all. Depending on the word being indexed and which system preferences are in effect (these are discussed in this chapter), the normalized form of a word may be the same as the original form. Also, the normalized form may comprise more than one spelling. For example, the normalized form of *Schoen* is both *Schoen* and *Schön*.

Oracle Text handles indexing of alternative word forms in the following ways:

- Alternate Spelling—indexing of alternative forms is enabled
- Base-Letter Conversion—accented letters are transformed into non-accented representations
- New German Spelling—reformed German spelling is accepted

Enable these features by specifying the appropriate attribute to the BASIC_LEXER. For instance, enable alternate spelling by specifying either GERMAN, DANISH, or SWEDISH for the ALTERNATE_SPELLING attribute. As an example, here is how to enable alternate spelling in German:

```
begin
ctx_ddl.create_preference('GERMAN_LEX', 'BASIC_LEXER');
ctx_ddl.set_attribute('GERMAN_LEX', 'ALTERNATE_SPELLING', 'GERMAN');
end;
```

To disable alternate spelling, use the CTX DDL.UNSET ATTRIBUTE procedure as follows:

```
begin
ctx_ddl.unset_attribute('GERMAN_LEX', 'ALTERNATE_SPELLING');
end:
```

Oracle Text converts query terms to their normalized forms before lookup. As a result, users can query words with either spelling. If *Schoen* has been indexed as both *Schoen* and *Schön*, a query with *Schön* returns documents containing either form.

This section contains the following topics.

- Alternate Spelling
- Base-Letter Conversion
- New German Spelling

18.1.1 Alternate Spelling

When Swedish, German, or Danish has more than one way of spelling a word, Oracle Text normally indexes the word in its original form; that is, as it appears in the source document.

When Alternate Spelling is enabled, Oracle Text indexes words in their normalized form. So, for example, *Schoen* is indexed both as *Schoen* and as *Schön*, and a query on *Schoen* will return documents containing either spelling. (The same is true of a query on *Schön*.)

To enable Alternate Spelling, set the BASIC_LEXER attribute ALTERNATE_SPELLING to GERMAN, DANISH, or SWEDISH. See "BASIC LEXER" for more information.

18.1.2 Base-Letter Conversion

Besides alternative spelling, Oracle Text also handles base-letter conversions. With base-letter conversions enabled, letters with umlauts, acute accents, cedillas, and the like are converted to their basic forms for indexing, so *fiancé* is indexed both as *fiancé* and as *fiance*, and a query of *fiancé* returns documents containing either form.

To enable base-letter conversions, set the BASIC_LEXER attribute BASE_LETTER to YES. See "BASIC LEXER" for more information.

When Alternate Spelling is also enabled, Base-Letter Conversion may need to be overridden to prevent unexpected results. See "Overriding Alternative Spelling Features" for more information.

Generic Versus Language-Specific Base-Letter Conversions

The BASE_LETTER_TYPE attribute affects the way base-letter conversions take place. It has two possible values: GENERIC or SPECIFIC.

The GENERIC value is the default and specifies that base letter transformation uses one transformation table that applies to all languages.

The SPECIFIC value means that a base-letter transformation that has been specifically defined for your language will be used. This enables you to use accent-sensitive searches for words in your own language, while ignoring accents that are from other languages.

For example, both the GENERIC and the Spanish SPECIFIC tables will transform \acute{e} into e. However, they treat the letter $\~{n}$ distinctly. The GENERIC table treats $\~{n}$ as an n with an accent (actually, a tilde), and so transforms $\~{n}$ to n. The Spanish SPECIFIC table treats $\~{n}$ as a separate letter of the alphabet, and thus does not transform it.

18.1.3 New German Spelling

In 1996, new spelling rules for German were approved by representatives from all Germanspeaking countries. For example, under the spelling reforms, *Potential* becomes *Potenzial*, *Schiffahrt* becomes *Schifffahrt*, and *schneuzen* becomes *schnäuzen*.

When the BASIC_LEXER attribute NEW_GERMAN_SPELLING is set to YES, then a CONTAINS query on a German word that has both new and traditional forms will return documents matching both

forms. For example, a query on *Potential* returns documents containing both *Potential* and *Potenzial*. The default setting is NO.



Under reformed German spelling, many words traditionally spelled as one word, such as *soviel*, are now spelled as two (*so viel*). Currently, Oracle Text does not make these conversions, nor conversions from two words to one (for example, *weh tun* to *wehtun*).

The case of the transformed word is determined from the first two characters of the word in the source document; that is, *schiffahrt* becomes *schifffahrt*, *schiffahrt* becomes *schifffahrt*, and *schifffahrt* becomes *schifffahrt*.

As many new German spellings include hyphens, it is recommended that users choosing NEW GERMAN SPELLING define hyphens as printjoins.

See "BASIC_LEXER" for more information on setting this attribute.

18.2 Overriding Alternative Spelling Features

Even when alternative spelling features have been specified by lexer preference, it is possible to override them.

You can override base-letter conversion when Alternate Spelling is used, to prevent characters with alternate spelling forms, such as \ddot{u} , \ddot{o} , and \ddot{a} , from also being transformed to the base letter forms.

Overriding Base-Letter Transformations with Alternate Spelling

Transformations caused by turning on alternate_spelling are performed before those of base letter, which can sometimes cause unexpected results when both are enabled.

When Alternate Spelling is enabled, Oracle Text converts two-letter forms to single-letter forms (for example, ue to \ddot{u}), so that words can be searched in both their base and alternate forms. Therefore, with Alternate Spelling enabled, a search for *Schoen* will return documents with both *Schoen* and *Schön*.

However, when Base-letter Transformation is also enabled, the \ddot{u} in *Schlüssel* is transformed into a u, producing the non-existent word (in German, anyway) *Schlussel*, and the word is indexed in all three forms.

To prevent this secondary conversion, set the OVERRIDE BASE LETTER attribute to TRUE.

OVERRIDE_BASE_LETTER only affects letters with umlauts; accented letters, for example, are still transformed into their base forms.

For more on BASE_LETTER, see "Base-Letter Conversion".

18.3 Alternative Spelling Conventions

The following sections show the alternative spelling substitutions used by Oracle Text.

German Alternate Spelling Conventions

- Danish Alternate Spelling Conventions
- Swedish Alternate Spelling Conventions

18.3.1 German Alternate Spelling Conventions

The German alphabet is the English alphabet plus the additional characters: ä ö ü ß. Table 18-1 lists the alternate spelling conventions Oracle Text uses for these characters.

Table 18-1 German Alternate Spelling Conventions

Character	Alternate Spelling Substitution
ä	ae
ü	ue
Ö	oe
Ä	AE
Ü	UE
Ö	OE
ß	ss

18.3.2 Danish Alternate Spelling Conventions

The Danish alphabet is the Latin alphabet without the w, plus the special characters: $\emptyset \not \approx \mathring{a}$. Table 18-2 lists the alternate spelling conventions Oracle Text uses for these characters.

Table 18-2 Danish Alternate Spelling Conventions

Character	Alternate Spelling Substitution	
æ	ae	
Ø	oe	
å	aa	
Æ	AE	
Ø	OE	
Å	AA	

18.3.3 Swedish Alternate Spelling Conventions

The Swedish alphabet is the English alphabet without the *w*, plus the additional characters: å ä ö. Table 18-3 lists the alternate spelling conventions Oracle Text uses for these characters.

Table 18-3 Swedish Alternate Spelling Conventions

Alternate Spelling Convention	
ae	
aa	
oe	
AE	



Table 18-3 (Cont.) Swedish Alternate Spelling Conventions

Character	Alternate Spelling Convention
Å	AA
Ö	OE



A

Oracle Text Result Tables

This appendix describes the structure of the result tables used to store the output generated by the procedures in the CTX QUERY, CTX DOC, and CTX THES packages.

The following topics are discussed in this appendix:

- CTX_QUERY Result Tables
- CTX_DOC Result Tables
- CTX_THES Result Tables and Data Types

A.1 CTX_QUERY Result Tables

For the CTX_QUERY procedures that return results, tables for storing the results must be created before the procedure is called. The tables can be named anything, but must include columns with specific names and data types.

This section describes the following types of result tables, and their required columns:

- EXPLAIN Table
- HFEEDBACK Table

A.1.1 EXPLAIN Table

This section describes the EXPLAIN table.

- EXPLAIN Table Structure
- EXPLAIN Table Operation Column Values
- EXPLAIN Table OPTIONS Column Values

A.1.1.1 EXPLAIN Table Structure

Table A-1 describes the structure of the table to which CTX_QUERY.EXPLAIN writes its results.

Table A-1 EXPLAIN Result Table

Column Name	Datatype	Description
EXPLAIN_ID	VARCHAR2(30)	The value of the explain_id argument specified in the FEEDBACK call.
ID	NUMBER	A number assigned to each node in the query execution tree. The root operation node has ID =1. The nodes are numbered in a top-down, left-first manner as they appear in the parse tree.

Table A-1 (Cont.) EXPLAIN Result Table

Column Name	Datatype	Description
PARENT_ID	NUMBER	The ID of the execution step that operates on the output of the ID step. Graphically, this is the parent node in the query execution tree. The root operation node (ID =1) has PARENT_ID = 0.
OPERATION	VARCHAR2(30)	Name of the internal operation performed. Refer to Table A-2 for possible values.
OPTIONS	VARCHAR2(30)	Characters that describe a variation on the operation described in the OPERATION column. When an OPERATION has more than one OPTIONS associated with it, OPTIONS values are concatenated in the order of processing. See Table A-3 for possible values.
OBJECT_NAME	VARCHAR2(80)	Section name, wildcard term, weight, or threshold value or term to lookup in the index.
POSITION	NUMBER	The order of processing for nodes that all have the same PARENT_ID. The positions are numbered in ascending order starting at 1.
CARDINALITY	NUMBER	Reserved for future use. You should create this column for forward compatibility.

A.1.1.2 EXPLAIN Table Operation Column Values

Table A-2 shows the possible values for the OPERATION column of the EXPLAIN table.

Table A-2 EXPLAIN Table OPERATION Column

Operation Value	Query Operator	Equivalent Symbol
ABOUT	ABOUT	n/a
ACCUMULATE	ACCUM	,
AND	AND	&
COMPOSITE	(none)	n/a
EQUIVALENCE	EQUIV	=
MINUS	MINUS	-
NEAR	NEAR	;
NOT	NOT	~
NO_HITS	(no hits will result from this query)	n/a
OR	OR	
PHRASE	(a phrase term)	n/a
SECTION	(section)	n/a
THRESHOLD	>	>
WEIGHT	*	*
WITHIN	within	n/a
WORD	(a single term)	n/a

A.1.1.3 EXPLAIN Table OPTIONS Column Values

Table A-3 lists the possible values for the OPTIONS column of the EXPLAIN table.

Table A-3 EXPLAIN Table OPTIONS Column

Options Value	Description
(\$)	Stem
(?)	Fuzzy
(!)	Soundex
(T)	Order for ordered Near.
(F)	Order for unordered Near.
(n)	A number associated with the max_span parameter for the Near operator.
[9]	Indicates that index_stems is set and query is using token_type 9.

A.1.2 HFEEDBACK Table

This section describes the HFEEDBACK table.

- HFEEDBACK Table Structure
- HFEEDBACK Table Operation Column Values
- HFEEDBACK Table OPTIONS Column Values
- CTX_FEEDBACK_TYPE

A.1.2.1 HFEEDBACK Table Structure

Table A-4 describes the table to which CTX_QUERY.HFEEDBACK writes its results.

Table A-4 HFEEDBACK Results Table

Column Name	Datatype	Description
FEEDBACK_ID	VARCHAR2(30)	The value of the feedback_id argument specified in the HFEEDBACK call.
ID	NUMBER	A number assigned to each node in the query execution tree. The root operation node has ID =1. The nodes are numbered in a top-down, left-first manner as they appear in the parse tree.
PARENT_ID	NUMBER	The ID of the execution step that operates on the output of the ID step. Graphically, this is the parent node in the query execution tree. The root operation node (ID =1) has PARENT_ID = 0.
OPERATION	VARCHAR2(30)	Name of the internal operation performed. Refer to Table A-5 for possible values.



Table A-4 (Cont.) HFEEDBACK Results Table

Column Name	Datatype	Description
OPTIONS	VARCHAR2(30)	Characters that describe a variation on the operation described in the OPERATION column. When an OPERATION has more than one OPTIONS associated with it, OPTIONS values are concatenated in the order of processing. See Table A-6 for possible values.
OBJECT_NAME	VARCHAR2(80)	Section name, wildcard term, weight, threshold value or term to lookup in the index.
POSITION	NUMBER	The order of processing for nodes that all have the same PARENT_ID. The positions are numbered in ascending order starting at 1.
BT_FEEDBACK	CTX_FEEDBACK_TYPE	Stores broader feedback terms. See Table A-7.
PT_FEEDBACK	CTX_FEEDBACK_TYPE	Stores related feedback terms. See Table A-7.
NT_FEEDBACK	CTX_FEEDBACK_TYPE	Stores narrower feedback terms. See Table A-7.

A.1.2.2 HFEEDBACK Table Operation Column Values

Table A-5 shows the possible values for the OPERATION column of the HFEEDBACK table.

Table A-5 HFEEDBACK Results Table OPERATION Column

Operation Value	Query Operator	Equivalent Symbol
ABOUT	ABOUT	(none)
ACCUMULATE	ACCUM	,
AND	AND	&
EQUIVALENCE	EQUIV	=
MINUS	MINUS	-
NEAR	NEAR	;
NOT	NOT	~
OR	OR	
SECTION	(section)	
TEXT	word or phrase of a text query	
THEME	word or phrase of an ABOUT query	
THRESHOLD	>	>
WEIGHT	*	*
WITHIN	within	(none)

A.1.2.3 HFEEDBACK Table OPTIONS Column Values

Table A-6 lists the values for the OPTIONS column of the HFEEDBACK table.



Table A-6 HFEEDBACK Results Table OPTIONS Column

Options Value	Description
(T)	Order for ordered Near.
(F)	Order for unordered Near.
(n)	A number associated with the max_span parameter for the Near operator.

A.1.2.4 CTX FEEDBACK TYPE

The CTX_FEEDBACK_TYPE is a nested table of objects. This datatype is predefined in the CTXSYS schema. Use this type to define the columns BT_FEEDBACK, RT_FEEDBACK, and NT_FEEDBACK.

The nested table <code>CTX_FEEDBACK_TYPE</code> holds objects of type <code>CTX_FEEDBACK_ITEM_TYPE</code>, which is also predefined in the <code>CTXSYS</code> schema. This object is defined with three members and one method as follows:

Table A-7 CTX_FEEDBACK_ITEM_TYPE

CTX_FEEDBACK_ITEM_TYPE Members and Methods	Туре	Description
text	NUMBER	Feedback term.
cardinality	NUMBER	(reserved for future use.)
score	NUMBER	(reserved for future use.)

The SQL code that defines these objects is as follows:

See Also:

For an example of how to select from the HFEEDBACK table and its nested tables, refer to CTX_QUERY.HFEEDBACK in CTX_QUERY Package

A.2 CTX_DOC Result Tables

The CTX_DOC procedures return results stored in a table. Before calling a procedure, you must create the table. The tables can be named anything, but must include columns with specific names and data types.

This section describes the following result tables and their required columns:

- Filter Table
- Gist Table
- Highlight Table
- Markup Table
- Theme Table
- Token Table

A.2.1 Filter Table

A filter table stores one row for each filtered document returned by CTX_DOC.FILTER . Filtered documents can be plain text or HTML.

When you call CTX_DOC.FILTER for a document, the document is processed through the filter defined for the text column and the results are stored in the filter table you specify.

Filter tables can be named anything, but must include the following columns, with names and datatypes as specified:

Table A-8 FILTER Result Table

Column Name	Туре	Description
QUERY_ID	NUMBER	The identifier for the results generated by a particular call to CTX_DOC.FILTER (only populated when table is used to store results from multiple FILTER calls)
DOCUMENT	CLOB	Text of the document, stored in plain text or HTML.

A.2.2 Gist Table

A Gist table stores one row for each Gist/theme summary generated by CTX_DOC.GIST.

Gist tables can be named anything, but must include the following columns, with names and data types as specified:

Table A-9 Gist Table

Column Name	Туре	Description
QUERY_ID	NUMBER	Query ID.
POV	VARCHAR2(80)	Document theme. Case depends of how themes were used in document or represented in the knowledge base.
		POV has the value of GENERIC for the document GIST.
GIST	CLOB	Text of Gist or theme summary, stored as plain text



A.2.3 Highlight Table

A highlight table stores offset and length information for highlighted terms in a document. This information is generated by CTX_DOC.HIGHLIGHT . Highlighted terms can be the words or phrases that satisfy a word or an ABOUT query.

If a document is formatted, the text is filtered into either plain text or HTML and the offset information is generated for the filtered text. The offset information can be used to highlight query terms for the same document filtered with CTX_DOC.FILTER.

Highlight tables can be named anything, but must include the following columns, with names and datatypes as specified:

Table A-10 Highlight Table

Column Name	Туре	Description
QUERY_ID	NUMBER	The identifier for the results generated by a particular call to CTX_DOC.HIGHLIGHT (only populated when table is used to store results from multiple HIGHLIGHT calls)
OFFSET	NUMBER	The position of the highlight in the document, relative to the start of document which has a position of 1.
LENGTH	NUMBER	The length of the highlight.

A.2.4 Markup Table

A markup table stores documents in plain text or HTML format with the query terms in the documents highlighted by markup tags. This information is generated when you call CTX_DOC.MARKUP .

Markup tables can be named anything, but must include the following columns, with names and datatypes as specified:

Table A-11 Markup Table

Column Name	Туре	Description
QUERY_ID	NUMBER	The identifier for the results generated by a particular call to CTX_DOC.MARKUP (only populated when table is used to store results from multiple MARKUP calls)
DOCUMENT	CLOB	Marked-up text of the document, stored in plain text or HTML format

A.2.5 Theme Table

A theme table stores one row for each theme generated by CTX_DOC.THEMES. The value stored in the THEME column is either a single theme phrase or a string of parent themes, separated by colons.

Theme tables can be named anything, but must include the following columns, with names and data types as specified:

Table A-12 Theme Table

Column Name	Туре	Description
QUERY_ID	NUMBER	Query ID
THEME	VARCHAR2 (2000)	Theme phrase or string of parent themes separated by colons (:).
WEIGHT	NUMBER	Weight of theme phrase relative to other theme phrases for the document.

A.2.6 Token Table

A token table stores the text tokens for a document as output by the CTX_DOC.TOKENS procedure. Token tables can be named anything, but must include the following columns, with names and data types as specified.

Table A-13 Token Table

Column Name	Туре	Description
QUERY_ID	NUMBER	The identifier for the results generated by a particular call to CTX_DOC.HIGHLIGHT (only populated when table is used to store results from multiple HIGHLIGHT calls)
TOKEN	VARCHAR2 (255)	The token string in the text.
OFFSET	NUMBER	The position of the token in the document, relative to the start of document which has a position of 1.
LENGTH	NUMBER	The character length of the token.
THES_TOKENS	VARCHAR2(4000)	Synonyms or broader terms generated using the thesaurus for the token in TOKEN column. These are separated using colons.

A.3 CTX_THES Result Tables and Data Types

The CTX_THES expansion functions such as BT, NT, and SYN can return the expansions in a table of type EXP TAB. Specify the name of your table with the restab argument.

EXP_TAB Table Type

A.3.1 EXP_TAB Table Type

The EXP TAB table type is a table of rows of type EXP REC.

The EXP REC and EXP TAB types are defined as follows in the CTXSYS schema:

```
type exp_rec is record (
    xrel varchar2(12),
    xlevel number,
    xphrase varchar2(256)
);

type exp_tab is table of exp_rec index by binary_integer;
```

When you call a thesaurus expansion function and specify restab, the system returns the expansion as an $\texttt{EXP_TAB}$ table. Each row in this table is of type $\texttt{EXP_REC}$ and represents a word or phrase in the expansion. Table A-14 describes the fields in $\texttt{EXP_REC}$:

Table A-14 EXP_TAB Table Type (EXP_REC)

EXP_REC Field	Description
xrel	The xrel field contains the relation of the term to the input term (for example, 'SYN', 'PT', 'RT', and so on). The xrel value is PHRASE when the input term appears in the expansion. For translations, the xrel value is the language.
xlevel	The xlevel field is the level of the relation. This is used mainly when xrel is a hierarchical relation (BT*/NT*).
	The xlevel field is 0 when xrel is PHRASE.
	The xlevel field is 2 for translations of synonyms under TRSYN.
	The $xlevel$ field is 1 for operators that are not hierarchical, such as PT and RT.
xphrase	The xphrase is the related term. This includes a qualifier in parentheses, if one exists for the related term. Compound terms are not de-compounded.



B

Oracle Text Supported Document Formats

Oracle Text uses the HTML export technology of Oracle Outside In for automatic filtering. This appendix provides tables with the document and graphic file formats supported by the automatic AUTO FILTER filtering technology for this release.

This appendix contains the following topics:

- About Document Filtering Technology
- Supported Document Formats

See Also:

"AUTO_FILTER" for information on using AUTO_FILTER

B.1 About Document Filtering Technology

The automatic filtering technology in Oracle Text enables you to convert documents to HTML for document presentation with the CTX DOC package.

To use automatic filtering for indexing and DML processing, you must specify the AUTO_FILTER object in your filter preference.

To use automatic filtering technology for converting documents to HTML with the CTX_DOC package, you need not use the AUTO FILTER indexing preference.

This section contains these topics:

- Latest Updates for Patch Releases
- Restrictions on Format Support
- Supported Platforms for AUTO FILTER Document Filtering Technology
- Filtering on PDF Documents and Security Settings
- PDF Filtering Limitations
- Environment Variables
- General Limitations

B.1.1 Latest Updates for Patch Releases

The supported platforms and formats listed in this appendix apply for this release. These supported formats are updated for patch releases.

B.1.2 Restrictions on Format Support

The formats listed in this appendix are those formats recognized by AUTO_FILTER. Recognizing a format does not necessarily mean that text can be extracted from it. For example, a scanned document is usually an image and AUTO_FILTER does not perform optical character recognition. Similarly, text cannot be extracted for indexing from multimedia file types.

Password-protected documents and documents with password-protected content are not supported by the AUTO FILTER filter.

For other limitations, see "Supported Document Formats" concerning specific document types.

B.1.3 Supported Platforms for AUTO_FILTER Technology

These are the supported platforms for automatic filtering technology that enables you to convert documents to HTML with the CTX DOC package.

Supported Platform	Details
Windows Server (x86 64-bit)	 Windows Server 2008 x64 Standard, Enterprise, and Datacenter Editions (64-bit Extended Systems) Windows Server 2013 x64 Standard, Datacenter, and Essentials editions Windows Server 2016 x64 Standard, Datacenter, and Essentials editions Windows Server 2019 x64 Standard, Datacenter, and Essentials editions
HP-UX	HP-UX (PA-RISC 64-bit) 11.iHP-UX (Itanium 64) 11i
IBM AIX	 IBM AIX on POWER Systems (64-bit) 7.1 IBM AIX on POWER Systems (64-bit) 7.x
Red Hat Linux	 Red Hat Linux (x86-64) Red Hat Enterprise Linux (RHEL) 6, 7, 8 Red Hat Linux (z-series, s390-64) Red Hat Enterprise Linux (RHEL) 6, 7, 8 Red Hat Linux (PPC-64) Red Hat Enterprise Linux (RHEL) 6, 7, 8 Red Hat Linux (Arm-64) Red Hat Enterprise Linux (RHEL), 6, 7, 8
SuSE Linux	 SuSE Linux (X86-64) 12, 15 SuSE Linux (z-series, s390-64) 12, 15 SuSE Linux (PPC-64) 12, 15 SuSE Linux (Arm-64) 12, 15
Sun Solaris	Sun Solaris (SPARC 64 bit) 11.xSun Solaris (X86-64) 11.x

Note:

Some of these platforms may not be supported by the Oracle Database.



B.1.4 Filtering on PDF Documents and Security Settings

A PDF document can have different levels of security settings as follows:

Table B-1 AUTO_FILTER Behavior with PDF Security Settings

Security Level	Description	PDF Version	Encryption	AUTO_FILTER Support Level
Level 1	Requires a password for opening the document.	1.2+	40 bit RC4	Not supported.
Level 1	Requires a password for opening the document.	1.4+	128 bit RC4	Not supported.
Level 1	Requires a password for opening the document.	1.5+	128 bit RC4	Not supported.
Level 1	Requires a password for opening the document.	1.6+	128 bit AES	Not supported.
Level 1	Requires a password for opening the document.	1.7+	256 bit AES	Not supported.
Level 2	Disallows user printing of the document.	1.2+	40 bit RC4	Supported.
Level 2	Disallows user printing of the document.	1.4+	128 bit RC4	Supported.
Level 2	Disallows user printing of the document.	1.5+	128 bit RC4	Supported.
Level 2	Disallows user printing of the document.	1.6+	128 bit AES	Not supported.
Level 2	Disallows user printing of the document.	1.7+	256 bit AES	Not supported.
Level 3	Disallows user modification or change of the document.	1.2+	40 bit RC4	Supported.
Level 3	Disallows user modification or change of the document.	1.4+	128 bit RC4	Supported.
Level 3	Disallows user modification or change of the document.	1.5+	128 bit RC4	Supported.
Level 3	Disallows user modification or change of the document.	1.6+	128 bit RC4	Not supported.
Level 3	Disallows user modification or change of the document.	1.7+	256 bit AES	Not supported.
Level 4	Disallows the user from copying or extracting content from the document.	1.2+	40 bit RC4	Supported.
Level 4	Disallows the user from copying or extracting content from the document.	1.4+	128 bit RC4	Supported.
Level 4	Disallows the user from copying or extracting content from the document.	1.5+	128 bit RC4	Supported.
Level 4	Disallows the user from copying or extracting content from the document.	1.6+	128 bit AES	Not supported.
Level 4	Disallows the user from copying or extracting content from the document.	1.7+	256 bit AES	Not supported.



Note:

Starting with Oracle Database 21c, older encryption and hashing algorithms are deprecated.

The deprecated algorithms for <code>DBMS_CRYPTO</code> and native network encryption include MD4, MD5, DES, 3DES, and RC4-related algorithms as well as 3DES for Transparent Data Encryption (TDE). Removing older, less secure cryptography algorithms prevents accidental use of these algorithms. To meet your security requirements, Oracle recommends that you use more modern cryptography algorithms, such as the Advanced Encryption Standard (AES).

B.1.5 PDF Filtering Limitations

The following limitations apply when filtering PDF files:

- Multi-byte PDFs are supported, provided the PDF document is created using Character ID-keyed (CID) fonts, predefined CJK CMap files, or ToUnicode font encodings, and the document does not contain embedded fonts.
- Embedded fonts in a PDF document are not filtered correctly. They are usually displayed using the question mark (?) replacement character.
- Hyperlinks in a PDF are not active when displayed in a browser or a viewing window.
- Annotations, such as notes, sound, or movies, are not supported.

B.1.6 Environment Variables

No environment variables need to be set by the user.

B.1.7 General Limitations

AUTO FILTER filter technology has the following limitations:

- Any ASCII characters less then 0x20 (decimal 32) are converted to hexadecimal numbers.
- Files larger than 2GB are not handled.

B.2 Supported Document Formats

Document filtering is used for indexing, processing data manipulation language (DML), and converting documents into HTML with the $\mathtt{CTX_DOC}$ package. These are the document formats that Oracle Text supports for filtering.

- Archive File Format
- Database Formats
- E-Book Formats
- Email Formats
- Graphic Formats (Raster and Vector Image)
- Multimedia Formats
- Other Formats



- Presentation Formats
- Spreadsheet Formats
- Text and Markup Formats
- Word Processing and Desktop Publishing Formats



These lists do not represent the complete list of formats that Oracle Text is able to process. The <code>USER_FILTER</code> and <code>PROCEDURE_FILTER</code> enable Oracle Text to process <code>any</code> document format, provided an external filter exists that can filter to some textual format like plain-text, HTML, XML, and so forth.

B.2.1 Archive File Format

These are the archive formats that Oracle Text supports. When filtering an archive file, all the contents of the files inside the archive are exported to a single output file. This also includes the contents of all subfolders and files inside the archive file.

Table B-2 Supported Archive File Formats

Archive Format	Version
7z (BZIP2 and split archives not supported)	-
7z Self Extracting .exe (BZIP2 and split archives not supported)	-
LZA Self Extracting Compress	-
LZH Compress	-
Microsoft Office Binder	-
Microsoft Cabinet (CAB)	95 – 97
RAR	1.5, 2.0, 2.9, 5.x, 6.x
Self-extracting .exe	-
UNIX Compress	-
UNIX GZip	-
UNIX Tar	-
Uuencode	-
Zip	PKZip
Zip	WinZip
Zip	Zip64

B.2.2 Database Formats

These are the database formats that Oracle Text supports for filtering.

Format	Version	
DataEase	4.x	
DBase	III, IV, V, X, X1	
First Choice DB	Through 3.0	
Framework DB	3.0	
Microsoft Access (text only)	1.0, 2.0, 95–2019	
Microsoft Access Report Snapshot (File ID only)	2000 – 2003	
Microsoft Works DB for DOS	2.0	
Microsoft Works DB for Macintosh	2.0	
Microsoft Works DB for Windows	3.0, 4.0	
Paradox for DOS	2.0 - 4.0	
Paradox for Windows	1.0	
Q&A Database	Through 2.0	
R:BASE	R:BASE 5000	
R:BASE	R:BASE System V	
Reflex	2.0	
SmartWare II DB	1.02	

B.2.3 E-Book Formats

These are the supported e-book file formats that are viewable on e-book readers.

Format	Version
EPUB (File ID only)	-
MOBI (File ID only)	-

B.2.4 Email Formats

These are the formats that Oracle Text supports for email messages, encodings, attachments, Multipurpose Internet Mail Extensions (MIME) formats, and so on.

Format	Version
Apple Mail Message (EMLX)	2.0
Encoded mail messages	MHT
Encoded mail messages	Multi Part Alternative
Encoded mail messages	Multi Part Digest
Encoded mail messages	Multi Part Mixed
Encoded mail messages	Multi Part News Group
Encoded mail messages	Multi Part Signed
Encoded mail messages	TNEF
EML with Digital Signature	SMIME



Format	Version
IBM Lotus Notes Domino XML Language DXL	8.5
IBM Lotus Notes NSF (File ID)	7.x, 8.x
IBM Lotus Notes NSF (Win32, Win64, Linux x86-32 and Oracle Solaris 32-bit only with Notes Client or Domino Server)	8.x
MBOX Mailbox	RFC 822
Microsoft Outlook Message (MSG)	97 – 2013
Microsoft Outlook Express (EML)	-
Microsoft Outlook Forms Template (OFT)	97 – 2013
Microsoft Outlook OLM	2011 for Mac
Microsoft Outlook OST	97 – 2013
Microsoft Outlook PST	97 – 2013
Microsoft Outlook PST (Mac)	2001
MSG with Digital Signature	SMIME

MIME Support Notes

The following formats are supported:

- MIME formats
 - EML
 - MHT (Web Archive)
 - NWS (Newsgroup single-part and multi-part)
 - Simple Text Mail (defined in RFC 2822)
- TNEF format
- MIME encodings, including
 - base64 (defined in RFC 1521)
 - binary (defined in RFC 1521)
 - binhex (defined in RFC 1741)
 - btoa
 - quoted-printable (defined in RFC 1521)
 - utf-7 (defined in RFC 2152)
 - uue
 - xxe
 - venc

In addition, the body of a message can be encoded in several ways. The following encodings are supported:

- HTML
- RTF
- TNEF



- Text/enriched (defined in RFC 1523)
- Text/richtext (defined in RFC1341)
- Embedded mail message (defined in RFC 822) this is handled as a link to a new message

The attachments of a MIME message can be stored in many formats. Oracle Corporation processes all attachment types that its technology supports.

B.2.5 Graphic Formats (Raster and Vector Image)

The graphic formats that the AUTO_FILTER filter recognizes ensure that indexing a text column containing any of these formats produces no error. Formats are categorized as either embedded graphics or standalone graphics.

Embedded graphics are inserted or referenced within a document.

- Table B-3
- Table B-4



The AUTO FILTER filter cannot extract textual information from graphics.

Table B-3 Supported Raster Image Formats for AUTO_FILTER Filter

Format	Version
Adobe Photoshop	4.0
Adobe Photoshop PSD (File ID only)	-
Adobe Photoshop	CS1 – 6, CC 2014 - 2018
CALS Raster (GP4)	Type I
CALS Raster (GP4)	Type II
Computer Graphics Metafile	ANSI
Computer Graphics Metafile	CALS
Computer Graphics Metafile	NIST
Encapsulated PostScript (EPS)	TIFF header Only
GEM Image (Bitmap)	-
Graphics Interchange Format (GIF)	-
IBM Graphics Data Format (GDF)	1.0
IBM Picture Interchange Format	1.0
JBIG2	Graphic Embeddings in PDF
JFIF (JPEG not in TIFF format)	-
JPEG	-
JPEG 2000	JP2
Kodak Flash Pix	-
Kodak Photo CD	1.0



Table B-3 (Cont.) Supported Raster Image Formats for AUTO_FILTER Filter

Format	Version
Lotus PIC	-
Lotus Snapshot	-
Macintosh PICT	BMP only
Macintosh PICT2	BMP only
MacPaint	-
Microsoft Windows Bitmap	-
Microsoft Windows Cursor	-
Microsoft Windows Icon	-
OS/2 Bitmap	-
OS/2 Warp Bitmap	-
Paint Shop Pro (Win32 only)	5.0, 6.0
PC Paintbrush (PCX)	-
PC Paintbrush DCX (multi-page PCX)	-
Portable Bitmap (PBM)	-
Portable Graymap PGM	-
Portable Network Graphics (PNG)	-
Portable Pixmap (PPM)	-
Portable Arbitrary Map (PAM) (File ID only)	-
Progressive JPEG	-
StarOffice Draw	6.x - 9.0
Sun Raster	-
TIFF	Group 5 & 6
TIFF CCITT	Group 3 & 4
TruVision TGA (Targa)	2.0
WebP (File ID only)	-
Word Perfect Graphics	1.0
JT Image (File ID only)	8.0, 9.0, 10.0
WBMP wireless graphics format	-
X-Windows Bitmap	x10 compatible
X-Windows Dump	x10 compatible
X-Windows Pixmap	x10 compatible
WordPerfect Graphics	2.0 – 10.0

Table B-4 Supported Vector Image Formats for AUTO_FILTER Filter

Graphics Format	Version
Adobe FrameMaker (MIF only)	3.0 - 6.0
Adobe Illustrator Postscript	Level 2
Adobe Illustrator	4.0 – 7.0



Table B-4 (Cont.) Supported Vector Image Formats for AUTO_FILTER Filter

Graphics Format	Version
Adobe Illustrator (PDF Preview only)	9.0, CS1 - 6
Adobe Illustrator XMP	CS1 - 6
Adobe InDesign XMP	CS1 - 6
Adobe InDesign Interchange (XMP only)	-
Adobe PDF	1.0 - 1.7 (Acrobat 1 - 10)
Adobe PDF Package	1.7 (Acrobat 8 – 10)
Adobe PDF Portfolio	1.7 (Acrobat 8 – 10)
Ami Draw	SDW
AutoCAD Drawing	2.5, 2.6
AutoCAD Drawing	9.0 – 14.0
AutoCAD Drawing	2000i – 2015, 2016 – 2021
AutoShade Rendering	2
Corel Draw	2.0 – 9.0 and X7
Corel Draw Clipart	5.0, 7.0
Enhanced Metafile (EMF)	-
Escher graphics	-
FrameMaker Graphics (FMV)	3.0 – 5.0
Gem File (Vector)	-
Harvard Graphics Chart DOS	2.0 – 3.0
Harvard Graphics for Windows	-
Hewlett Packard Graphics Language (HPGL)	2.0
IGES Drawing	5.1 – 5.3
Micrografx Designer (DRW)	Through 3.1
Micrografx Designer (DFS)	6.0
Micrografx Draw (DRW)	Through 4.0
Microsoft XPS (Text only)	-
Novell PerfectWorks Draw	2
OpenOffice Draw	1.1 – 3.0
Oracle Open Office Draw	3.x
SVG (processed as XML, not rendered)	-
Visio (Page Preview mode WMF/EMF)	4.0
Visio	5.0 - 2010
Visio (text only)	2013
Visio XML VSX (File ID only)	2007
Windows Metafile (WMF)	-

B.2.6 Multimedia Formats

This table lists the multimedia formats that are recognized by ${\tt AUTO_FILTER}.$

Recognizing a format does not necessarily mean that text can be extracted from it. Also, the file name and file header information are not indexed. A scanned document is usually an image, and <code>AUTO_FILTER</code> does not perform optical character recognition. Similarly, text cannot be extracted for indexing from multimedia file types.

Format	Version
AVI (Metadata only)	-
DICOM (File ID only)	-
Flash (text extraction only)	6.x, 7.x, Lite
Flash (File ID only)	9, 10
Real Media (File ID only)	-
MP3 (ID3 metadata only)	-
MPEG-1 Audio layer 3 V ID3 v1 (Metadata only)	-
MPEG-1 Audio layer 3 V ID3 v2 (Metadata only)	-
MPEG-1 Video V 2 (File ID only)	-
MPEG-1 Video V 3 (File ID only)	-
MPEG-2 Audio (File ID only)	-
MPEG-4 (Metadata only)	-
MPEG-7 (Metadata only)	-
QuickTime (Metadata only)	-
Windows Media ASF (Metadata only)	-
Windows Media DVR-MS (Metadata only)	-
Windows Media Audio WMA (Metadata only)	-
Windows Media Playlist (File ID only)	-
Windows Media Video WMV (Metadata only)	-
WAV (Metadata only)	-
Apple HEIF (File ID only)	-
WebM (File ID only)	-

B.2.7 Other Formats

Format	Version
AOL Messenger (File ID only)	7.3
Microsoft InfoPath (File ID only)	2007
Microsoft Live Messenger (via XML filter)	10.0
Microsoft Office Theme files (File ID only	2007 - 2019
Microsoft OneNote (text only)	2007 - 2019
Microsoft Project (table view only)	98 – 2010
Microsoft Windows Compiled Help (File ID only)	.chm
Microsoft Windows DLL (File ID only)	.dll
Microsoft Windows Executable (File ID only)	.exe.com
Microsoft Windows Explorer Command (File ID only)	.scf

Format	Version
Microsoft Windows Help (File ID only)	.hlp
Microsoft Windows Shortcut (File ID only)	.lnk
Trillian Text Log File (via text filter)	4.2
Trillian XML Log File (File ID only)	4.2
TrueType Font (File ID only)	Ttf, ttc
vCalendar	2.1
vCard	2.1
Yahoo Messenger	6.x – 8

B.2.8 Presentation Formats

These are the presentation file formats that Oracle Text supports for filtering.

Format	Version
Apple iWork Keynote (text and PDF preview)	09, 2014, 2020
Harvard Graphics Presentation DOS	3.0
IBM Lotus Symphony Presentations	1.x
Kingsoft WPS Presentation	2010
LibreOffice Impress	4.x, 5.x, 6.x
Lotus Freelance	1.0 – Millennium 9.8
Lotus Freelance for OS/3	2
Lotus Freelance for Windows	95, 97, SmartSuite 9.8
Microsoft PowerPoint for Macintosh	4.0 – 2016, 2019
Microsoft PowerPoint for Windows	3.0 – 2016, 2019
Microsoft PowerPoint for Windows Slideshow	2007 – 2019
Microsoft PowerPoint for Windows Template	2007 – 2019
Novell Presentations	3.0, 7.0
OpenOffice Impress	1.1, 3.0, 4.x
Oracle Open Office Impress	3.x
StarOffice Impress	5.2 – 9.0
Strict Open XML –Presentation (File ID only)	2013, 2019
WordPerfect Presentations	5.1 – X7
Advanced Function Presentation (AFP) (File ID only)	-

B.2.9 Spreadsheet Formats

These are the spreadsheet file formats that Oracle Text supports for filtering.

Format	Version
Apple iWork Numbers (text and PDF preview)	09



Format	Version
Apple iWork Numbers (File ID only)	2014, 2020
Enable Spreadsheet	3.0 – 4.5
First Choice SS	Through 3.0
Framework SS	3.0
IBM Lotus Symphony Spreadsheets	1.x
Kingsoft WPS Spreadsheets	2010
LibreOffice Calc	4.x
Lotus 1-2-3	Through Millennium 9.8
Lotus 1-2-3 Charts for DOS and Windows	Through 5.0
Lotus 1-2-3 for OS/2	2.0
Microsoft Excel Charts	2.x - 2007
Microsoft Excel for Macintosh	98 – 2011
Microsoft Excel for Windows	3.0 – 2019
Microsoft Excel for Windows (text only)	2003 XML
Microsoft Excel for Windows (.xlsb)	2007 - 2019 (Binary)
Microsoft Works SS for DOS	2.0
Microsoft Works SS for Macintosh	2.0
Microsoft Works SS for Windows	3.0, 4.0
Multiplan	4.0
Novell PerfectWorks Spreadsheet	2.0
OpenOffice Calc	1.1 – 3.0
Oracle Open Office Calc	3.x
PFS: Plan	1.0
Quattro Pro for DOS	Through 5.0
Quattro Pro for Windows	Through X7
SmartWare Spreadsheet	-
SmartWare II SS	1.02
StarOffice Calc	5.2 – 9.0
SuperCalc	5.0
Symphony	Through 2.0
VP-Planner	1.0

B.2.10 Text and Markup Formats

These are the formats for text and markup versions of documents that Oracle Text supports.

Format	Version
ANSI Text	7 and 8 bit
ASCII Text	7 and 8 bit
Ami Pro for OS2	-



Format	Version
Ami Pro for Windows	2.0, 3.0
Apple iWork Pages (text and PDF preview)	09
Apple iWork Pages (File ID only)	2014, 2020
DEC DX	Through 4.0
DEC DX Plus	4.0, 4.1
Enable Word Processor	3.0 – 4.5
First Choice WP	1.0, 3.0
Framework WP	3.0
Hangul	97 – 2010
IBM DCA/FFT	-
IBM DisplayWrite	2.0 - 5.0
IBM Writing Assistant	1.01
Ichitaro	5.0, 6.0, 8.0 – 13.0, 2004, 2010, 2013
JustWrite	Through 3.0
Kingsoft WPS Writer	2010
Legacy	1.1
LibreOffice Writer	4.x
Lotus Manuscript	Through 2.0
Lotus WordPro (text only)	9.7, 96 – Millennium 9.8
MacWrite II	1.1
Mass 11	Through 8.0
Microsoft Publisher (File ID only)	2003 - 2016
Microsoft Word for DOS	4.0 - 6.0
Microsoft Word for Macintosh	4.0 – 6.0, 98 – 2011
Microsoft Word for Windows	1.0 – 2016, 2019
Microsoft Word for Windows (text only)	2003 XML
DOS character set	-
EBCDIC	-
HTML (HTML5 advanced elements are limited to those typically found in HTML based emails.)	1.0 – 5.0
IBM DCA/RFT	-
Macintosh character set	-
Rich Text Format (RTF)	-
Unicode Text	3.0, 4.0
UTF-8	· -
Wireless Markup Language	-
XML (Text only)	-
XHTML (File ID only)	1.0
XML Localization Interchange File Format (File ID only)	-
XML Forms Data Format (File ID only)	-



B.2.11 Word Processing and Desktop Publishing Formats

These are the formats for word processing and desktop publishing handled by Oracle Text filters.

Format	Version				
Adobe FrameMaker (MIF only)	3.0 - 6.0				
Adobe Illustrator Postscript	Level 2				
Ami	-				
Ami Pro for OS2	-				
Ami Pro for Windows	2.0, 3.0				
Apple iWork Pages (Text and PDF preview)	09				
Apple iWork Pages (File ID only)	2014, 2020				
DEC DX	Through 4.0				
DEC DX Plus	4.0, 4.1				
Enable Word Processor	3.0 - 4.5				
First Choice WP	1.0, 3.0				
Framework WP	3.0				
Hangul	97 – 2010				
IBM DCA/FFT	-				
IBM DisplayWrite	2.0 - 5.0				
IBM Writing Assistant	1.01				
Ichitaro	5.0, 6.0, 8.0 – 13.0, 2004, 2013				
JustWrite	Through 3.0				
Kingsoft WPS Writer	2010				
Legacy	1.1				
LibreOffice Writer	4.x				
Lotus Manuscript	Through 2.0				
Lotus WordPro (text only)	9.7, 96 – Millennium 9.8				
MacWrite II	1.1				
Mass 11	Through 8.0				
Microsoft Word for DOS	4.0 - 6.0				
Microsoft Word for Macintosh	4.0 – 6.0, 98 – 2011				
Microsoft Word for Windows	1.0 – 2016, 2019				
Microsoft Word for Windows (text only via XML filter)	2003 XML				
Microsoft Word for Windows	98-J				
Microsoft WordPad	-				
Microsoft Works WP for DOS	2.0				
Microsoft Works WP for Macintosh	2.0				
Microsoft Works WP for Windows	3.0, 4.0				
Microsoft Write for Windows	1.0 – 3.0				
MultiMate	Through 4.0				



Format	Version				
MultiMate Advantage	2.0				
Navy DIF	-				
Nota Bene	3.0				
Novell PerfectWorks Word Processor	2.0				
OfficeWriter	4.0 - 6.0				
OpenOffice Writer	1.1 – 3.0				
Oracle Open Office Writer	3.x				
PC File Doc	5.0				
PFS: Write	А, В				
Professional Write for DOS	1.0, 2.0				
Professional Write Plus for Windows	1.0				
Q&A Write	2.0, 3.0				
Samna Word IV	1.0 – 3.0				
Samna Word IV+	-				
Signature	1.0				
SmartWare II WP	1.02				
Sprint	1.0				
StarOffice Writer	5.2 – 9.0				
Strict Open XML –Document (file ID only)	2013, 2016, 2019				
Total Word	1.2				
Wang IWP	Through 2.6				
WordMarc Composer	-				
WordMarc Composer+	-				
WordMarc Word Processor	-				
WordPerfect for DOS	4.2				
WordPerfect for Macintosh	1.02 – 3.1				
WordPerfect for Windows	5.1 – X7				
WordStar 2000 for DOS	1.0 – 3.0				
Wordstar for DOS	3.0 – 7.0				
Wordstar for Windows	1.0				
XyWrite	Through III+				



C

Text Loading Examples for Oracle Text

This appendix provides examples of how to load text into a text column, and the structure of ctxload import files. This appendix contains these topics:

- SQL INSERT Example
- SQL*Loader Example
- Structure of ctxload Thesaurus Import File

C.1 SQL INSERT Example

A simple way to populate a text table is to create a table with two columns, id and text, using CREATE TABLE and then use the INSERT statement to load the data. This example makes the id column the primary key, which is optional. The text column is VARCHAR2:

```
create table docs (id number primary key, text varchar2(80));
```

To populate the text column, use the INSERT statement as follows:

```
insert into docs values(1, 'this is the text of the first document'); insert into docs values(12, 'this is the text of the second document');
```

C.2 SQL*Loader Example

The following example shows how to use SQL*Loader to load mixed format documents from the operating system to a BLOB column. The example has two steps:

- Creating the Table
- Issuing the SQL*Loader Command

The SQL*Loader command reads the control file and loads data into table

See Also:

For a complete discussion on using SQL*Loader, see Oracle Database Utilities

C.2.1 Creating the Table

This example loads a table articles formatted created as follows:

```
CREATE TABLE articles_formatted (
ARTICLE_ID NUMBER PRIMARY KEY ,
AUTHOR VARCHAR2(30),
FORMAT VARCHAR2(30),
PUB_DATE DATE,
TITLE VARCHAR2(256),
```



```
TEXT BLOB
```

The article_id column is the primary key. Documents are loaded in the text column, which is of type BLOB.

C.2.2 Issuing the SQL*Loader Command

The following command starts the loader, which reads the control file LOADER1.DAT:

sqlldr userid=demo/password control=loader1.dat log=loader.log

- Example Control File: loader1.dat
- Example Data File: loader2.dat

C.2.2.1 Example Control File: loader1.dat

This SQL*Loader control file defines the columns to be loaded and instructs the loader to load the data line by line from <code>loader2.dat</code> into the <code>articles_formatted</code> table. Each line in <code>loader2.dat</code> holds a comma-delimited list of fields to be loaded.

```
-- load file example
load data
INFILE 'loader2.dat'
INTO TABLE articles_formatted
APPEND
FIELDS TERMINATED BY ','
(article_id SEQUENCE (MAX,1),
author CHAR(30),
format,
pub_date SYSDATE,
title,
ext_fname FILLER CHAR(80),
text LOBFILE(ext_fname) TERMINATED BY EOF)
```

This control file instructs the loader to load data from <code>loader2.dat</code> to the <code>articles_formatted</code> table in the following way:

- The ordinal position of the line describing the document fields in loader2.dat is written to the article id column.
- 2. The first field on the line is written to author column.
- 3. The second field on the line is written to the format column.
- 4. The current date given by SYSDATE is written to the pub date column.
- 5. The title of the document, which is the third field on the line, is written to the title column.
- 6. The name of each document to be loaded is read into the <code>ext_fname</code> temporary variable, and the actual document is loaded in the <code>text</code> BLOB column:

C.2.2.2 Example Data File: 10ader2.dat

This file contains the data to be loaded into each row of the table, articles formatted.

Each line contains a comma-delimited list of the fields to be loaded in articles_formatted. The last field of every line names the file to be loaded in to the text column:

```
Ben Kanobi, plaintext, Kawasaki news article, .. /sample docs/kawasaki.txt,
Joe Bloggs, plaintext, Java plug-in, .. /sample docs/javaplugin.txt,
John Hancock, plaintext, Declaration of Independence, .. / sample_docs/indep.txt,
M. S. Developer, Word7, Newsletter example, .. / sample docs/newsletter.doc,
M. S. Developer, Word7, Resume example, ../sample_docs/resume.doc,
X. L. Developer, Excel7, Common example, .. / sample docs/common.xls,
X. L. Developer, Excel7, Complex example, .. / sample docs/solvsamp.xls,
Pow R. Point, Powerpoint7, Generic presentation, .. /sample docs/generic.ppt,
Pow R. Point, Powerpoint7, Meeting presentation, .. /sample docs/meeting.ppt,
Java Man, PDF, Java Beans paper,../sample_docs/j_bean.pdf,
Java Man, PDF, Java on the server paper, .. /sample docs/j svr.pdf,
Ora Webmaster, HTML, Oracle home page, .. / sample_docs/oramnu97.html,
Ora Webmaster, HTML, Oracle Company Overview, .. /sample_docs/oraoverview.html,
John Constable, GIF, Laurence J. Ellison: portrait, .. /sample docs/larry.gif,
Alan Greenspan, GIF, Oracle revenues: Graph,../sample docs/oragraph97.gif,
Giorgio Armani, GIF, Oracle Revenues: Trend, .. /sample docs/oratrend.gif,
```

C.3 Structure of ctxload Thesaurus Import File

This section discusses the structure of the ctxload thesaurus import file in the following topics.

- Import File Format
- Alternate Hierarchy Structure
- Usage Notes for Terms in Import Files
- Usage Notes for Relationships in Import Files
- Examples of Import Files

C.3.1 Import File Format

The import file must use the following format for entries in the thesaurus:

```
phrase
BT broader_term
NT narrower term1
NT narrower term2
NT narrower_termN
BTG broader_term
NTG narrower term1
NTG narrower term2
NTG narrower termN
BTP broader term
NTP narrower term1
NTP narrower term2
NTP narrower termN
BTI broader_term
NTI narrower_term1
NTI narrower_term2
NTI narrower_termN
SYN synonym1
SYN synonym2
```

```
SYN synonymN

USE synonym1 or SEE synonym1 or PT synonym1

RT related_term1

RT related_term2

...

RT related_termN

SN text

language_key: term
```

phrase

is a word or phrase that is defined as having synonyms, broader terms, narrower terms, or related terms.

In compliance with ISO-2788 standards, a TT marker can be placed before a phrase to indicate that the phrase is the top term in a hierarchy; however, the TT marker is not required. In fact, ctxload ignores TT markers during import.

A top term is identified as any phrase that does not have a broader term (BT, BTG, BTP, or BTI).



The thesaurus query operators (SYN, PT, BT, BTG, BTP, BTI, NT, NTG, NTP, NTI, and RT) are reserved words and, thus, cannot be used as phrases in thesaurus entries.

BT, BTG, BTP, BTI broader termN

are the markers that indicate <code>broader_termN</code> is a broader (generic|partitive|instance) term for <code>phrase</code>.

broader_termN is a word or phrase that conceptually provides a more general description or category for phrase. For example, the word *elephant* could have a broader term of *land mammal*.

NT, NTG, NTP, NTI narrower_termN

are the markers that indicate narrower_termN is a narrower (generic|partitive|instance) term for phrase.

If phrase does not have a broader (generic|partitive|instance) term, but has one or more narrower (generic|partitive|instance) terms, phrase is created as a top term in the respective hierarchy (in an Oracle Text thesaurus, the BT/NT, BTG/NTG, BTP/NTP, and BTI/NTI hierarchies are separate structures).

narrower_termN is a word or phrase that conceptually provides a more specific description for phrase. For example, the word *elephant* could have a narrower terms of *indian elephant* and *african elephant*.

SYN synonymN

is a marker that indicates <code>phrase</code> and <code>synonymN</code> are synonyms within a synonym ring. <code>synonymN</code> is a word or phrase that has the same meaning for <code>phrase</code>. For example, the word dog could have a synonym of canine.





Synonym rings are not defined explicitly in Oracle Text thesauri. They are created by the transitive nature of synonyms.

USE SEE PT synonym1

are markers that indicate phrase and synonym1 are synonyms within a synonym ring (similar to SYN).

The markers USE, SEE or PT also indicate synonym1 is the preferred term for the synonym ring. Any of these markers can be used to define the preferred term for a synonym ring.

Note:

If the user-defined thesaurus is to be used for compiling the knowledge base, then you must specify the preferred term when a synonym ring is declared. Use one of the keywords USE, SEE, or PT to specify which synonym to use when reporting query matches. Only one term can be a preferred term.

Not using one of these keywords may result in the failure to return results defined by a word's synonym. When compiling two or more thesauri that declare elements of the same synonym ring, the preferred term must be the same in both files, which ensures that only one word is defined as the preferred word in a synonym ring.

RT related termN

is the marker that indicates related_termN is a related term for phrase. related_termN is a word or phrase that has a meaning related to, but not necessarily synonymous with phrase. For example, the word dog could have a related term of wolf.



Related terms are not transitive. If a phrase has two or more related terms, the terms are related only to the parent phrase and not to each other.

SN text

is the marker that indicates the following text is a scope note (for example, comment) for the preceding entry.

language key term

term is the translation of phrase into the language specified by language key.

C.3.2 Alternate Hierarchy Structure

In compliance with thesauri standards, the load file supports formatting hierarchies (BT/NT, BTG/NTG, BTP, NTP, BTI/NTI) by indenting the terms under the top term and using NT (or NTG, NTP, NTI) markers that include the level for the term:

```
phrase
  NT1 narrower_term1
    NT2 narrower_term1.1
    NT2 narrower_term1.2
        NT3 narrower_term1.2.1
        NT3 narrower_term1.2.2
```



```
NT1 narrower_term2
. . .
NT1 narrower_termN
```

Using this method, the entire branch for a top term can be represented hierarchically in the load file.

C.3.3 Usage Notes for Terms in Import Files

The following conditions apply to the structure of the entries in the import file:

- Each entry (phrase, BT, NT, or SYN) must be on a single line followed by a newline character.
- Entries can consist of a single word or phrases.
- The maximum length of an entry (phrase, BT, NT, or SYN) is 255 bytes, not including the BT, NT, and SYN markers or the newline characters.
- Entries cannot contain parentheses or plus signs.
- Each line of the file that starts with a relationship (BT, NT, and so on) must begin with at least one space.
- A phrase can occur more than once in the file.
- Each phrase can have one or more narrower term entries (NT, NTG, NTP), broader term entries (BT, BTG, BTP), synonym entries, and related term entries.
- Each broader term, narrower term, synonym, and preferred term entry must start with the appropriate marker and the markers must be in capital letters.
- The broader terms, narrower terms, and synonyms for a phrase can be in any order.
- Homographs must be followed by parenthetical disambiguators everywhere they are used.

```
For example: cranes (birds), cranes (lifting equipment)
```

- Compound terms are signified by a plus sign between each factor (for example, buildings + construction).
- Compound terms are allowed only as synonyms or preferred terms for other terms, never as terms by themselves, or in hierarchical relations.
- Terms can be followed by a scope note (SN), total maximum length of 2000 bytes, on subsequent lines.
- Multi-line scope notes are allowed, but require an SN marker on each line of the note.

Example of Incorrect SN usage:

```
VIEW CAMERAS

SN Cameras with through—the lens focusing and a range of movements of the lens plane relative to the film plane
```

Example of Correct SN usage:

```
VIEW CAMERAS

SN Cameras with through-the lens focusing and a

SN range of movements of the lens plane relative

SN to the film plane
```

• Multi-word terms cannot start with reserved words (for example, *use* is a reserved word, so *use other door* is not an allowed term; however, *use* is an allowed term).

C.3.4 Usage Notes for Relationships in Import Files

The following conditions apply to the relationships defined for the entries in the import file:

- related term entries must follow a phrase or another related term entry
- related term entries start with one or more spaces, the RT marker, followed by white space, then the related term on the same line
- multiple related terms require multiple RT markers

Example of incorrect RT usage:

```
MOVING PICTURE CAMERAS
RT CINE CAMERAS
TELEVISION CAMERAS
```

Example of correct RT usage:

```
MOVING PICTURE CAMERAS
RT CINE CAMERAS
RT TELEVISION CAMERAS
```

Terms are allowed to have multiple broader terms, narrower terms, and related terms

C.3.5 Examples of Import Files

This section provides three examples of correctly formatted thesaurus import files.

- Example 1 (Flat Structure)
- Example 2 (Hierarchical)
- Example 3

C.3.5.1 Example 1 (Flat Structure)

```
cat
 SYN feline
 NT domestic cat
NT wild cat
BT mammal
mammal
BT animal
domestic cat
NT Persian cat
NT Siamese cat
wild cat
NT tiger
tiger
NT Bengal tiger
dog
BT mammal
 NT domestic dog
NT wild dog
SYN canine
domestic dog
NT German Shepard
wild dog
NT Dingo
```

C.3.5.2 Example 2 (Hierarchical)

```
animal
  NT1 mammal
             NT3 domestic cat
                  NT4 Persian cat
                  NT4 Siamese cat
             NT3 wild cat
                 NT4 tiger
                      NT5 Bengal tiger
        NT2 dog
            NT3 domestic dog
                 NT4 German Shepard
             NT3 wild dog
                  NT4 Dingo
cat
SYN feline
dog
SYN canine
```

C.3.5.3 Example 3

```
35MM CAMERAS
BT MINIATURE CAMERAS
CAMERAS
BT OPTICAL EQUIPMENT
NT MOVING PICTURE CAMERAS
NT STEREO CAMERAS
LAND CAMERAS
USE VIEW CAMERAS
VIEW CAMERAS
SN Cameras with through-the lens focusing and a range of SN movements of the lens plane relative to the film plane UF LAND CAMERAS
BT STILL CAMERAS
```



Oracle Text Multilingual Features

This Appendix describes the multilingual features of Oracle Text. The following topics are discussed:

- Introduction
- Indexing
- Querying
- Supplied Stop Lists
- Knowledge Base
- Multilingual Features Matrix

D.1 Introduction

This appendix summarizes the main multilingual features for Oracle Text.

For a complete list of Oracle Globalization Support languages and character set support, refer to the *Oracle Database Globalization Support Guide*.



Oracle Text does not support the NLS_COMP and NLS_SORT parameters. Search results generated from Oracle Text are independent from values of those parameters.

In Oracle Database 12c Release 2 (12.2), an Oracle Text index cannot be created on a column with a declared collation other than <code>BINARY</code>, <code>USING_NLS_COMP</code>, <code>USING_NLS_SORT</code> or <code>USING_NLS_SORT_CS</code>. For all the supported collations, the Oracle Text behavior is the same.

D.2 Indexing

The following sections describe the multilingual indexing features:

- Multilingual Features for Text Index Types
- Lexer Types
- Basic Lexer Features
- Multi Lexer Features
- World Lexer Features

D.2.1 Multilingual Features for Text Index Types

The following sections describes the supported multilingual features for the Oracle Text index types.

- CONTEXT Index Type
- CTXCAT Index Type
- CTXRULE Index Type

See Also:

"Lexer Types" for a description of available lexers

D.2.1.1 CONTEXT Index Type

The CONTEXT index type fully supports multilingual features, including use of the language and character set columns.

The following lexers are supported:

- AUTO_LEXER
- BASIC_LEXER
- MULTI LEXER
- USER_LEXER
- WORLD LEXER
- CHINESE LEXER
- CHINESE_VGRAM_LEXER
- JAPANESE LEXER
- JAPANESE_VGRAM_LEXER
- KOREAN MORPH LEXER

D.2.1.2 CTXCAT Index Type

CTXCAT supports the multilingual features of the BASIC_LEXER with the exception of indexing themes, and supports the following additional lexers:

- USER LEXER
- WORLD LEXER

CTXCAT also supports the following lexers:

- CHINESE_LEXER
- CHINESE_VGRAM_LEXER
- JAPANESE_LEXER
- JAPANESE VGRAM LEXER
- KOREAN_MORPH_LEXER



Note:

The Oracle Text indextype CTXCAT is deprecated with Oracle Database 23ai. The indextype itself, and it's operator CTXCAT, can be removed in a future release. Both CTXCAT and the use of CTXCAT grammar as an alternative grammar for CONTEXT queries is deprecated. Instead, Oracle recommends that you use the CONTEXT indextype, which can provide all the same functionality, except that it is not transactional. Near-transactional behavior in CONTEXT can be achieved by using SYNC (ON COMMIT) or, preferably, SYNC (EVERY [time-period]) with a short time period.

CTXCAT was introduced when indexes were typically a few megabytes in size. Modern, large indexes, can be difficult to manage with CTXCAT. The addition of index sets to CTXCAT can be achieved more effectively by the use of FILTER BY and ORDER BY columns, or SDATA, or both, in the CONTEXT indextype. CTXCAT is therefore rarely an appropriate choice. Oracle recommends that you choose the more efficient CONTEXT indextype.

D.2.1.3 CTXRULE Index Type

The CTXRULE index type supports the multilingual features of the BASIC_LEXER including ABOUT and STEM operators. It also supports Japanese, Chinese, and Korean (when used with the SVM CLASSIFIER).

D.2.2 Lexer Types

Oracle Text supports the indexing of different languages by enabling you to choose a lexer in the indexing process. The specified lexer determines the languages you can index.

Table D-1 Oracle Text Lexer Types

Lexer	Supported Languages	
AUTO_LEXER	Automatically identifies the language being indexed by examining the content, and applies suitable options (including stemming) for that language. Works best where each document contains a single-language, and has at least a couple of paragraphs of text to aid identification.	
BASIC_LEXER	Extracts tokens from text in languages, such as English and most of the western European languages that use space-delimited words.	
MULTI_LEXER	Indexes tables containing documents of different languages such as English, German, and Japanese.	
CHINESE_VGRAM	Extracts tokens from Chinese text.	
CHINESE_LEXER	Extracts tokens from Chinese text. This lexer offers the following benefits over the CHINESE_VGRAM lexer:	
	 Generates a smaller index Better query response time Generates real world tokens resulting in better query precision Supports stop words 	
JAPANESE_VGRAM	Extracts tokens from Japanese text.	



Table D-1 (Cont.) Oracle Text Lexer Types

Lexer	Supported Languages
JAPANESE_LEXER	Extracts tokens from Japanese text. This lexer offers the following advantages over the <code>JAPANESE_VGRAM</code> lexer:
	Generates smaller index
	Better query response time
	 Generates real world tokens resulting in better precision
KOREAN_MORPH_LEXER	Extracts tokens from Korean text.
USER_LEXER	Indexes a particular language.
WORLD_LEXER	Indexes tables containing documents of different languages; autodetects languages in a document

D.2.3 Basic Lexer Features

The following features are supported with the <code>BASIC_LEXER</code> preference. Enable these features with attributes of the <code>BASIC_LEXER</code>. Features such as alternate spelling, composite, and base letter can be enabled together for better search results.

- Theme Indexing
- Alternate Spelling
- Base Letter Conversion
- Composite
- Index stems

D.2.3.1 Theme Indexing

Enables the indexing and subsequent querying of document concepts with the ABOUT operator with CONTEXT index types. These concepts are derived from the Oracle Text knowledge base. This feature is supported for English and French.

This feature is not supported with CTXCAT index types.

D.2.3.2 Alternate Spelling

This feature enables you to search on alternate spellings of words. For example, with alternate spelling enabled in German, a query on *gross* returns documents that contain *groß* and *gross*.

This feature is supported in German, Danish, and Swedish.

Additionally, German can be indexed according to both traditional and reformed spelling conventions.



"Alternate Spelling" and "New German Spelling".



D.2.3.3 Base Letter Conversion

This feature enables you to query words with or without diacritical marks such as tildes, accents, and umlauts. For example, with a Spanish base-letter index, a query of *energia* matches documents containing both *energía* and *energia*.

This feature is supported for English and all other supported whitespace delimited languages. In English and French, you can use the basic lexer to enable theme indexing.



"Base-Letter Conversion"

D.2.3.4 Composite

This feature enables you to search for words that contain the specified term as a sub-composite. You must use the stem (\$) operator.

For example, in German, a query of \$register finds documents that contain Bruttoregistertonne and Registertonne.

You can use this feature for all languages that are supported for the <code>INDEX_STEMS</code> attribute of <code>BASIC LEXER</code>.

Related Topics

- BASIC_LEXER Attributes
 These are the attributes supported for the BASIC LEXER component.
- BASIC_LEXER Language Support
 Oracle Text installs language data files for English by default. You can download data files for all other supported languages on demand from My Oracle Support.

D.2.3.5 Index Stems

This feature enables you to specify a stemmer for stem indexing.

Tokens are stemmed to a single base form at index time in addition to the normal forms. Specifying index stems enables better query performance for stem queries, for example \$computed.

You can use this feature for all languages that are supported for the INDEX_STEMS attribute of BASIC_LEXER.

Related Topics

- BASIC LEXER Attributes
 - These are the attributes supported for the BASIC LEXER component.
- BASIC LEXER Language Support

Oracle Text installs language data files for English by default. You can download data files for all other supported languages on demand from My Oracle Support.



D.2.4 Multi Lexer Features

The MULTI_LEXER lexer enables you to index a column that contains documents of different languages. During indexing Oracle Text examines the language column and switches in the language-specific lexer to process the document. Define the lexer preferences for each language before indexing.

The multi lexer enables you to set different preferences for languages. For example, you can have composite set to TRUE for German documents and composite set to FALSE for Dutch documents.

D.2.5 World Lexer Features

Like MULTI_LEXER, the WORLD_LEXER lexer enables you to index documents that contain different languages. It automatically detects the languages of a document and, therefore, does not require you to create a language column in the base table.

WORLD_LEXER processes all database character sets and supports the Unicode 5.0 standard. For WORLD_LEXER to be effective with documents that use multiple languages, AL32UTF-8 or UTF8 Oracle character set encoding must be specified. This includes supplementary, or "surrogate-pair," characters.

Table D-2 and Table D-3 show the languages supported by WORLD_LEXER. This list may change as the Unicode standard changes, and in any case should not be considered exhaustive. (Languages are grouped by Unicode writing system, not by natural language groupings.)

Table D-2 Languages Supported by the World Lexer (Space-separated)

Language Group	Languages Include	
Arabic	Arabic, Farsi, Kurdish, Pashto, Sindhi, Urdu	
Armenian	Armenian	
Bengali	Assamese, Bengali	
Bopomofo	Hakka Chinese, Minnan Chinese	
Cyrillic	Over 50 languages, including Belorussian, Bulgarian, Macedonian, Moldavian, Russian, Serbian, Serbo-Croatian, Ukrainian	
Devenagari	Bhojpuri, Bihari, Hindi, Kashmiri, Marathi, Nepali, Pali, Sanskrit	
Ethiopic	Amharic, Ge'ez, Tigrinya, Tigre	
Georgian	Georgian	
Greek	Greek	
Gujarati	Gujarati, Kacchi	
Gurmukhi	Punjabi	
Hebrew	Hebrew, Ladino, Yiddish	
Kaganga	Redjang	
Kannada	Kanarese, Kannada	
Korean	Korean, Hanja Hangul	



Table D-2 (Cont.) Languages Supported by the World Lexer (Space-separated)

Language Group	Languages Include	
Latin	Afrikaans, Albanian, Basque, Breton, Catalan, Croatian, Czech, Danis Dutch, English, Esperanto, Estonian, Faeroese, Fijian, Finnish, Flemis French, Frisian, German, Hawaiian, Hungarian, Icelandic, Indonesian Irish, Italian, Lappish, Classic Latin, Latvian, Lithuanian, Malay, Malte Pinyin Mandarin, Maori, Norwegian, Polish, Portuguese, Provencal, Romanian, Rumanian, Samoan, Scottish Gaelic, Slovak, Slovene, Slovenian, Sorbian, Spanish, Swahili, Swedish, Tagalog, Turkish, Vietnamese, Welsh	
Malayalam	Malayalam	
Mongolian	Mongolian	
Oriya	Oriya	
Sinhalese, Sinhala	Pali, Sinhalese	
Syriac	Aramaic, Syriac	
Tamil	Tamil	
Telugu	Telugu	
Thaana	Dhiveli, Divehi, Maldivian	

Table D-3 Languages Supported by the World Lexer (Non-space-separated)

Language Group	Languages Include
Chinese	Cantonese, Mandarin, Pinyin phonograms
Japanese	Japanese (Hiragana, Kanji, Katakana)
Khmer	Cambodian, Khmer
Lao	Lao
Myanmar	Burmese
Thai	Thai
Tibetan	Dzongkha, Tibetan

Table D-4 shows languages not supported by the World Lexer.

Table D-4 Languages Not Supported by the World Lexer

Language Group	Languages Include
Buhid	Buhid
Canadian Syllabics	Blackfoot, Carrier, Cree, Dakhelh, Inuit, Inuktitut, Naskapi, Nunavik, Nunavut, Ojibwe, Sayisi, Slavey
Cherokee	Cherokee
Cypriot	Cypriot
Limbu	Limbu
Ogham	Ogham
Runic	Runic
Tai Le (Tai Lu, Lue, Dai Le)	Tai Le



Table D-4 (Cont.) Languages Not Supported by the World Lexer

Language Group	Languages Include
Ugaritic	Ugaritic
Yi	Yi
Yi Jang Hexagram	Yi Jang

D.3 Querying

Oracle Text supports the use of different query operators. Some operators can be set to behave in accordance with your language. This section summarizes the multilingual query features for these operators.

- Use the ABOUT operator to query on concepts. The system looks up concept information in the theme component of the index. This feature is supported for English and French with CONTEXT indexes only.
- The fuzzy operator enables you to search for words that have similar spelling to specified word. Oracle Text supports fuzzy for English, French, German, Italian, Dutch, Spanish, Portuguese, Japanese, Optical Character recognition (OCR), and automatic language detection.
- The stem operator enables you to search for words that have the same root as the specified term. For example, a stem of \$sing expands into a query on the words sang, sung, sing. The Oracle Text stemmer supports the following languages: English, French, Spanish, Italian, German, Japanese and Dutch.

D.4 Supplied Stoplists

By default, the system indexes text using the Oracle Text supplied stoplists that correspond to your database language.

A stoplist is a list of stopwords that do not get indexed. These are usually common words in a language, such as *this*, *that*, and *can* in English. By default, all such words are defined in the Oracle Text supplied stoplists. You can customize these stoplists or update the stopwords based on your requirements.

Supported Languages and Stoplists Location

The Oracle Text supplied stoplists contain a list of stopwords, which are provided as defaults for all BASIC_LEXER and AUTO_LEXER supported languages. These stopwords are automatically loaded during installation or upgrade for the chosen database language.

The default stoplists (along with other default preferences) are defined in the administration (SQL) files, which are located in the $\$ORACLE_HOME/ctx/admin$ directory. These SQL files are named drdefLANG.sql, where LANG specifies the language code. For example, the default stoplist for French (language code: f) is defined in the $\$ORACLE_HOME/ctx/admin/drdeff.sql$ file.

The source files for these default stoplists contain a list of stopwords, and are located in the <code>\$ORACLE_HOME/ctx/data/stoplist</code> directory. These source files are named <code>drstoplang.txt</code>, where <code>LANG</code> specifies the language code. The contents of the source files are the extracted terms from the <code>drdefLANG.sql</code> files.



For a list of all languages (and their language codes) in which default stoplists are supplied, see Multilingual Features Matrix.

How to Load Your Own Stoplists

By default, only one $\mathtt{drdef}\mathit{LANG}.sq1$ file is loaded during installation or upgrade based on the database language that you choose. You can call the $\mathtt{CTX_DDL}.\mathtt{LOAD_STOPLIST}$ procedure to customize your stoplist or modify the default list of stopwords.

Unlike CTX_DDL.ADD_STOPWORD (which adds a single stopword per call), CTX_DDL.LOAD_STOPLIST takes a source file of stopwords for your specified language (from \$ORACLE HOME/ctx/data/stoplist/drstopLANG.txt) and loads to your stoplist.

Related Topics

- LOAD STOPLIST
 - Use this procedure to load a source file of stopwords to your stoplist for the required language.
- Multilingual Features Matrix
 These are the multilingual features for all supported languages.

D.5 Knowledge Base

An Oracle Text knowledge base is a hierarchical tree of concepts used for theme indexing, ABOUT queries, and deriving themes for document services.

Oracle Text supplies knowledge bases in English and French only. These knowledge bases are installed by default.

You can extend theme functionality to languages other than English or French by loading your own knowledge base for any single byte white space delimited language, including Spanish.

D.6 Multilingual Features Matrix

These are the multilingual features for all supported languages.

Table D-5 Multilingual Features for Supported Languages

Language Name	Language Code	Alternate Spelling	Fuzzy Matching	Language- Specific Lexer	Default Stoplist	Stemming
Afrikaans	af	N/A	No	Yes	Yes	Yes
Arabic	ar	N/A	No	Yes	Yes	Yes
Basque	eu	N/A	No	Yes	Yes	Yes
Belarusian	be	N/A	No	Yes	Yes	Yes
Bokmal (Norwegian)	n	N/A	No	Yes	Yes	Yes
Bulgarian	bg	N/A	No	Yes	Yes	Yes
Catalan	ca	N/A	No	Yes	Yes	Yes
Simplified Chinese	zh-cn	N/A	No	Yes	Yes	Yes
Croatian	hr	N/A	No	Yes	Yes	Yes
Czech	cs	N/A	No	Yes	Yes	Yes
Danish	dk	Yes	No	Yes	Yes	Yes



Table D-5 (Cont.) Multilingual Features for Supported Languages

Language Name	Language Code	Alternate Spelling	Fuzzy Matching	Language- Specific Lexer	Default Stoplist	Stemming
Dutch	nl	N/A	Yes	Yes	Yes	Yes
English	us	N/A	Yes	Yes	Yes	Yes
Estonian	et	N/A	No	Yes	Yes	Yes
Finnish	sf	N/A	No	Yes	Yes	Yes
French	f	N/A	Yes	Yes	Yes	Yes
Galician	gl	N/A	No	Yes	Yes	Yes
German	d	Yes	Yes	Yes	Yes	Yes
Greek	el	N/A	No	Yes	Yes	Yes
Hebrew	iw	N/A	No	Yes	Yes	Yes
Hindi	hi	N/A	No	Yes	Yes	Yes
Hungarian	hu	N/A	No	Yes	Yes	Yes
Icelandic	is	N/A	No	Yes	Yes	Yes
Indonesian	in	N/A	No	Yes	Yes	Yes
Italian	i	N/A	Yes	Yes	Yes	Yes
Japanese	ja	N/A	Yes	Yes	Yes	Yes
Korean	ko	N/A	No	Yes	Yes	Yes
Latvian	lv	N/A	No	Yes	Yes	Yes
Lithuanian	It	N/A	No	Yes	Yes	Yes
Macedonian	mk	N/A	No	Yes	Yes	Yes
Malay	ms	N/A	No	Yes	Yes	Yes
Nynorsk (Norwegian)	nn	N/A	No	Yes	Yes	Yes
Persian (Farsi)	fa	N/A	No	Yes	Yes	Yes
Polish	pl	N/A	No	Yes	Yes	Yes
Portuguese	pt	N/A	Yes	Yes	Yes	Yes
Romanian	ro	N/A	No	Yes	Yes	Yes
Russian	ru	N/A	No	Yes	Yes	Yes
Slovak	sk	N/A	No	Yes	Yes	Yes
Slovenian	sl	N/A	No	Yes	Yes	Yes
Serbian	sr	N/A	No	Yes	Yes	Yes
Spanish	es	N/A	Yes	Yes	Yes	Yes
Swedish	sv	Yes	No	Yes	Yes	Yes
Thai	th	N/A	No	Yes	Yes	Yes
Traditional Chinese	zh-tw	N/A	No	Yes	Yes	Yes
Turkish	tr	N/A	No	Yes	Yes	Yes
Ukrainian	uk	N/A	No	Yes	Yes	Yes
Urdu	ur	N/A	No	Yes	Yes	Yes

F

The Oracle Text Scoring Algorithm

This appendix describes how Oracle Text calculates scoring for word queries, which is different from the way it calculates scores for ABOUT queries in English. Scoring is obtained using the SCORE operator.

This appendix contains these topics:

- Scoring Algorithm for Word Queries
- Word Scoring Example
- DML and Scoring Algorithm



"DEFINESCORE" and "DEFINEMERGE" for information about user-defined scoring

E.1 Scoring Algorithm for Word Queries

To calculate a relevance score for a returned document in a word query, Oracle Text uses an inverse frequency algorithm based on Salton's formula.

Inverse frequency scoring assumes that frequently occurring terms in a document set are noise terms, and so these terms are scored lower. For a document to score high, the query term must occur frequently in the document but infrequently in the document set as a whole.

The following table illustrates Oracle Text's inverse frequency scoring. The first column shows the number of documents in the document set, and the second column shows the number of terms in the document necessary to score 100.

This table assumes that only one document in the set contains the guery term.

Number of Documents in Document Set	Occurrences of Term in Document Needed to Score 100
1	34
5	20
10	17
50	13
100	12
500	10
1,000	9
10,000	7
100,000	6
1,000,000	5

Note that the score varies, depending on the set size. For example, if only one document in the set contains the query term, and there are five documents in the set, then the term must occur 20 times in the document to score 100. If 1,000,000 documents are in the set, then the term can occur only 5 times in the document to score 100.

E.2 Word Scoring Example

You have 5000 documents dealing with chemistry in which the term *chemical* occurs at least once in every document. The term *chemical* thus occurs frequently in the document set.

You have a document that contains 5 occurrences of *chemical* and 5 occurrences of the term *hydrogen*. No other document contains the term *hydrogen*. The term hydrogen thus occurs infrequently in the document set.

Because *chemical* occurs so frequently in the document set, its score for the document is lower with respect to *hydrogen*, which is infrequent is the document set as a whole. The score for *hydrogen* is therefore higher than that of *chemical*. This is so even though both terms occur 5 times in the document.



Even if the relatively infrequent term *hydrogen* occurred 4 times in the document, and *chemical* occurred 5 times in the document, the score for *hydrogen* might still be higher, because *chemical* occurs so frequently in the document set (at least 5000 times).

Inverse frequency scoring also means that adding documents that contain *hydrogen* lowers the score for that term in the document, and adding more documents that do not contain *hydrogen* raises the score.

E.3 DML and Scoring Algorithm

Because the scoring algorithm is based on the number of documents in the document set, inserting, updating or deleting documents in the document set is likely to change the score for any given term before and after DML.

If DML is heavy, you must optimize the index. Perfect relevance ranking is obtained by running a query right after optimizing the index.

If DML is light, Oracle Database still gives fairly accurate relevance ranking.

In either case, you must synchronize the index with CTX_DDL.SYNC_INDEX.





F

Oracle Text Views

This is a list of all the views provided by Oracle Text.

- CTX_ALEXER_DICTS
- CTX_AUTO_OPTIMIZE_INDEXES
- CTX_AUTO_OPTIMIZE_STATUS
- CTX_AUTOSYNC_JOBS
- CTX_AUTOSYNC_STATUS
- CTX_BACKGROUND_EVENTS
- CTX_USER_BACKGROUND_EVENTS
- CTX_CLASSES
- CTX_FILTER_BY_COLUMNS
- CTX_FILTER_CACHE_STATISTICS
- CTX INDEXES
- CTX_INDEX_ERRORS
- CTX_INDEX_OBJECTS
- CTX_INDEX_PARTITIONS
- CTX_INDEX_SETS
- CTX_INDEX_SET_INDEXES
- CTX_INDEX_SUB_LEXERS
- CTX_INDEX_SUB_LEXER_VALUES
- CTX_INDEX_VALUES
- CTX_OBJECTS
- CTX_OBJECT_ATTRIBUTES
- CTX OBJECT ATTRIBUTE LOV
- CTX_ORDER_BY_COLUMNS
- CTX_PARAMETERS
- CTX_PREFERENCES
- CTX_PREFERENCE_VALUES
- CTX_SECTIONS
- CTX_SECTION_GROUPS
- CTX SQES
- CTX_STOPLISTS
- CTX_STOPWORDS



- CTX SUB LEXERS
- CTX THESAURI
- CTX_THES_PHRASES
- CTX_TRACE_VALUES
- CTX USER ALEXER DICTS
- CTX_USER_AUTO_OPTIMIZE_INDEXES
- CTX_USER_AUTOSYNC_JOBS
- CTX USER AUTOSYNC STATUS
- CTX_USER_EXTRACT_POLICIES
- CTX_USER_EXTRACT_POLICY_VALUES
- CTX_USER_EXTRACT_RULES
- CTX USER EXTRACT STOP ENTITIES
- CTX_USER_EXTRACT_TYPE
- CTX_USER_FILTER_BY_COLUMNS
- CTX USER INDEXES
- CTX_USER_INDEX_ERRORS
- CTX_USER_INDEX_OBJECTS
- CTX_USER_INDEX_PARTITIONS
- CTX USER INDEX SETS
- CTX_USER_INDEX_SET_INDEXES
- CTX_USER_INDEX_SUB_LEXERS
- CTX_USER_INDEX_SUB_LEXER_VALS
- CTX USER INDEX VALUES
- CTX_USER_ORDER_BY_COLUMNS
- CTX_USER_PREFERENCES
- CTX_USER_PREFERENCE_VALUES
- CTX USER SECTIONS
- CTX_USER_SECTION_GROUPS
- CTX_USER_SQES
- CTX_USER_STOPLISTS
- CTX_USER_STOPWORDS
- CTX_USER_SUB_LEXERS
- CTX_USER_THESAURI
- CTX USER THES PHRASES
- CTX_VERSION
- ALL_DBMS_SEARCH_INDEXES
- ALL_DBMS_SEARCH_INDEX_SOURCES
- USER DBMS SEARCH INDEXES



USER_DBMS_SEARCH_INDEX_SOURCES

F.1 CTX_ALEXER_DICTS

This view displays all dictionaries created by any user. This view can be queried by CTXSYS.

Column Name	Туре	Description
ALD_OWNER	VARCHAR2(30)	Name of the dictionary owner
ALD_NAME	VARCHAR2(30)	Name of the dictionary
ALD_LANG	VARCHAR2(30)	Language of the dictionary

F.2 CTX_AUTO_OPTIMIZE_INDEXES

This view displays all the indexes that are registered for auto optimization. It can be queried by CTXSYS.

Column Name	Туре	Description
AOI_INDEX_OWNER	VARCHAR2(30)	Index owner
AOI_INDEX_NAME	VARCHAR2(30)	Index name
AOI_PARTITION_NAME	VARCHAR2(30)	Partition name

Note:

In Oracle Database Release 21c, the procedures <code>ADD_AUTO_OPTIMIZE</code> and <code>REMOVE_AUTO_OPTIMIZE</code>, and the views <code>CTX_AUTO_OPTIMIZE_INDEXES</code>, <code>CTX_USER_AUTO_OPTIMIZE_INDEXES</code> and <code>CTX_AUTO_OPTIMIZE_STATUS</code> are deprecated.

F.3 CTX_AUTO_OPTIMIZE_STATUS

This view displays the status of auto optimization jobs. It can be queried by CTXSYS.

Column Name	Туре	Description
AOS_TIMESTAMP	TIMESTAMP(6) WITH TIMEZONE	Time at which the auto optimization job started
AOS_STATUS	VARCHAR2(30)	Status of the auto optimization job
AOS_ERROR	VARCHAR2(4000)	Errors raised by the auto optimization job

Note:

In Oracle Database Release 21c, the procedures <code>ADD_AUTO_OPTIMIZE</code> and <code>REMOVE_AUTO_OPTIMIZE</code>, and the views <code>CTX_AUTO_OPTIMIZE_INDEXES</code>, <code>CTX_USER_AUTO_OPTIMIZE_INDEXES</code> and <code>CTX_AUTO_OPTIMIZE_STATUS</code> are deprecated.

F.4 CTX_AUTOSYNC_JOBS

The CTX_AUTOSYNC_JOBS view lists all background synchronization jobs that are owned by the CTXSYS user.

Column Name Type		Description
ASJ_JOB_NAME	VARCHAR2 (128)	Name of the synchronization job
ASJ_INDEX_NAME	VARCHAR2 (128)	Name of the Oracle Text index
ASJ_INDEX_ID	NUMBER (38)	Index ID
ASJ_INDEX_OWNER	VARCHAR2 (128)	Index owner
ASJ_PARTITION_NAME	VARCHAR2 (128)	Partition name
ASJ_PARTITION_ID	NUMBER	Partition ID
ASJ_START_DATE	TIMESTAMP(6) WITH TIME ZONE	Original start date of the synchronization job
ASJ_REPEAT_INTERVAL	VARCHAR2 (4000)	Repeat interval
ASJ_STATE	VARCHAR2(15)	Current state of the job
ASJ_RUN_COUNT	NUMBER	Number of times the synchronization job has run

Note:

Synchronization jobs are now owned by the CTXSYS user. You can not use the USER SCHEDULER JOBS view to view the background synchronization jobs for indexes.

F.5 CTX_AUTOSYNC_STATUS

The CTX_AUTOSYNC_STATUS view lists the status of each run of a background synchronization job that is owned by the CTXSYS user.

Column Name	Туре	Description
ASJ_JOB_NAME	VARCHAR2(1044)	Name of the synchronization job
ASJ_TIMESTAMP	IMESTAMP(6) WITH TIME ZONE	Date of the log entry
ASJ_STATUS	VARCHAR2(30)	Status of the synchronization job
ASJ_ERROR	VARCHAR2(4000)	Errors generated by the synchronization job

Note:

Synchronization jobs are now owned by the <code>CTXSYS</code> user. You can not use the <code>USER_SCHEDULER_JOBS</code> view to view the background synchronization jobs for indexes.



F.6 CTX_BACKGROUND_EVENTS

The CTX_BACKGROUND_EVENTS view displays historical information about the execution of background events for indexes with automatic maintenance. This view filters events for the SYS or CTXSYS user.

A similar view CTX_USER_BACKGROUND_EVENTS displays events for the current user based on the index owner.

The objects are represented by both their numbers and names. However, because the data is historical, it is possible that those objects may no longer exist. Thus, the object names may be \mathtt{NULL} , however the object numbers are never \mathtt{NULL} .

Column Name	Туре	Description
BGE OWNER#	NUMBER	Index owner number
BGE OWNER NAME	VARCHAR2 (128)	Index owner name
BGE TABLE#	NUMBER	Base table object number
BGE TABLE NAME	VARCHAR2 (128)	Base table name
 BGE INDEX#	NUMBER	Index object number
- BGE INDEX NAME	VARCHAR2 (128)	Index name
BGE_TABLE_PARTITION		Base table partition object number
BGE_TABLE_PARTITION _NAME	VARCHAR2 (128)	Base table partition name
BGE_INDEX_PARTITION #	NUMBER	Index partition object number
BGE_INDEX_PARTITION _NAME	VARCHAR2 (128)	Index partition name
BGE_EVENT_TYPE	VARCHAR2 (30)	Event type: NONE SYNC-Mapping Timeout (Sync-MT) SYNC-Mapping (Sync-M) SYNC-Ranges (Sync-R) SYNC-Scheduler (Sync-S) SYNC-Postings Serial (Sync-PS) SYNC-Postings Concurrent (Sync-PC) SYNC-Writer (Sync-W) SYNC-Cleanup batches (Sync-C) SYNC-Inspect (Sync-I) MONITOR EVENT Stats (EStat) EVENT Stats Clean up (EClean) OPTIMIZE-Scheduler Timeout (Opti-ST) OPTIMIZE-Scheduler (Opti-S)



Column Name	Туре	Description
BGE_EVENT_ITERATION	NUMBER	Event retries happen with an increased delay up to a maximum of 10 hr.
		Depending on the event type, the retries could go through a single event loop or through a longer loop. For example, if Sync-W fails, then it needs to go through Sync-C, Sync-S, SYNC-Postings (Sync-P) before it can reach Sync-W again.
		Event iterations track the number of times a longer loop has been retried, and event waits track single event delays.
BGE_EVENT_WAIT	NUMBER	Time in minutes for which the event waited to retry before it was scheduled. The delays increase each time the event fails and max out at 10 hr.
		The delays are 3 sec, 10 sec, 30 sec, 1 min, 3 min, 10 min, 30 min, 1 hr, 3 hr, and 10 hr.
BGE_WORKER_NUMBER	NUMBER	Internal worker number
BGE_BATCH_NUMBER	NUMBER	Internal SGA batch number
BGE_STATUS	VARCHAR2 (30)	 Event status: EMPTY: No status information RUNNING: Event is running DONE: Successful completion BUSY: Index or index partition is busy INTERRUPTED: Event is interrupted by DDL CLOSED: PDB is closed DROPPED: Index or index partition is dropped or altered FAILED: Event failed with an error FATAL: Worker process is terminated
BGE_START_TIME	TIMESTAMP(6) WITH TIMEZONE	Event processing start time
BGE_END_TIME	TIMESTAMP(6) WITH TIMEZONE	Event processing end time
BGE_DURATION	NUMBER	Event processing duration in seconds
BGE_ERROR_NUMBER	NUMBER	Top error number
BGE_ERROR_MESSAGE	VARCHAR2(4000)	Complete stack of error messages
BGE_NUM_ROWS	NUMBER	 Sync-M: Number of processed DMLs Sync-R: number of selected NEW ranges Sync-S: Number of READY ranges Sync-P: Number of processed documents (document count) Sync-W: Number of documents written Sync-C: Number of ranges deleted Sync-I: Number of events enqueued Monitor: Number of objects processed



Column Name	Туре	Description
BGE_FIRST_DOCID	NUMBER	 Sync-M: NEW range first DocID Sync-R: First DocID of the first READY range Sync-S: First DocID of the first READY range Sync-P: First DocID of the selected READY range Sync-W: First DocID of the WRITE range Opti-S: First DocID of the aggregated MERGE range
BGE_CURRENT_DOCID	NUMBER	Sync-P: Current DocID of the selected READY range (last processed)
BGE_LAST_DOCID	NUMBER	 Sync-M: NEW range last DocID Sync-R: Last DocID of the last READY range Sync-S: Last DocID of the last READY range Sync-P: Last DocID of the selected READY range Sync-W: Last DocID of the WRITE range Opti-S: Last DocID of the aggregated MERGE range
BGE_BATCH_WAIT	NUMBER	Sync-P: Time waited for batch in minutes
BGE_BATCH_SIZE	NUMBER	Sync-P: Batch memory size in bytesSync-W: Batch memory size in bytes
BGE_DOCUMENT_SIZE	NUMBER	Sync-R: Estimated per-document memory sizeSync-R: Average per-document memory size
BGE_TOKEN_COUNT	NUMBER	Sync-P: Token countSync-W: Token countOpti-S: Token count
BGE_RANGE_COUNT	NUMBER	 Sync-M: Number of deleted DocIDs Sync-R: Number of generated READY ranges Sync-P: Generated READY range count
BGE_RANGE_TYPE	NUMBER	 Sync-R: READY range type Sync-P: READY range type Ready range type: NA INITIAL RETRY

Related Topics

Oracle Text Application Developer's Guide

F.7 CTX_CLASSES

This view displays all the preference categories registered in the Text data dictionary. It can be queried by any user.

Column Name	Туре	Description
CLA_NAME	VARCHAR2(30)	Class name
CLA_DESCRIPTION	VARCHAR2(80)	Class description



F.8 CTX_FILTER_BY_COLUMNS

This view displays all FILTER BY columns registered in the Text data dictionary. It can be queried by any user.

Column Name	Туре	Description
FBC_INDEX_OWNER	VARCHAR2(30)	Index owner name
FBC_INDEX_NAME	VARCHAR2(30)	Index name
FBC_TABLE_OWNER	VARCHAR2(30)	Table owner name
FBC_TABLE_NAME	VARCHAR2(30)	Table name
FBC_COLUMN_NAME	VARCHAR2 (256)	Column name
FBC_COLUMN_TYPE	VARCHAR2(30)	Column type
FBC_SECTION_TYPE	VARCHAR2(30)	Section type
FBC_SECTION_NAME	VARCHAR2(30)	Section name
FBC_SECTION_ID	NUMBER	Section ID

F.9 CTX_FILTER_CACHE_STATISTICS

This view displays various statistics related to the query filter cache. This view can be queried by all users and it displays the statistics for all indexes.

Column Name	Туре	Description
FCS_INDEX_OWNER	VARCHAR2 (30)	Index owner name
FCS_INDEX_NAME	VARCHAR2(30)	Index name
FCS_PARTITION_NAME	VARCHAR2(30)	Index partition name
FCS_SIZE	NUMBER	Current size of the filter cache in bytes
FCS_ENTRIES	NUMBER	Number of queries for which the query results are cached in the filter cache
FCS_REQUESTS	NUMBER	Number of query requests to the filter cache
FCS_HITS	NUMBER	Number of query requests for which matches were found in the filter cache

Note:

Starting in Oracle Database Release 21c, CTXFILTERCACHE is deprecated, and also CTX_FILTER_CACHE_STATISTICS and QUERY_FILTER_CACHE_SIZE.

F.10 CTX INDEXES

This view displays all indexes that are registered in the Text data dictionary for the current user. It can be queried by CTXSYS.

Column Name	Туре	Description
IDX_CHARSET_COLUMN	VARCHAR2(256)	Name of the charset column in base table
IDX_DOCID_COUNT	NUMBER	Number of documents indexed
IDX_FORMAT_COLUMNS	VARCHAR2(256)	Name of the format column in base table
IDX_ID	NUMBER	Internal index ID
IDX_KEY_NAME	VARCHAR2(256)	Primary key column(s)
IDX_LANGUAGE_COLUMN	VARCHAR2(256)	Name of the language column in base table
IDX_NAME	VARCHAR2(30)	Name of index
IDX_OWNER	VARCHAR2(30)	Owner of index
IDX_STATUS	VARCHAR2(12)	Status
IDX_SYNC_INTERVAL	VARCHAR2 (2000)	Interval string required by scheduler job. Only meaningful for AUTOMATIC sync. Always null for MANUAL and ON COMMIT sync.
IDX_SYNC_JOBNAME	VARCHAR2 (50)	Scheduler job name for automatic sync. Only meaningful for AUTOMATIC sync and always null for other types of sync.
IDX_SYNC_MEMORY	VARCHAR2 (100)	Sync memory size. Only meaningful for ON COMMIT and AUTOMATIC types of sync. For MANUAL sync, this is always null.
IDX_SYNC_PARA_DEGREE	NUMBER	Degree of parallelism for sync. Only meaningful for the AUTOMATIC type of sync; always null for MANUAL and ON COMMIT syncs.
IDX_SYNC_TYPE	VARCHAR2 (20)	Type of synching: MANUAL, AUTOMATIC, or ON COMMIT
IDX_TABLE	VARCHAR2(30)	Table name
IDX_TABLE_OWNER	VARCHAR2(30)	Owner of table
IDX_TEXT_NAME	VARCHAR2(30)	Text column name
IDX TYPE	VARCHAR2(7)	Type of index: CONTEXT, CTXCAT, or CTXRULE

F.11 CTX_INDEX_ERRORS

This view displays the DML errors and is queryable by ${\tt CTXSYS}.$

Column Name	Туре	Description
ERR_INDEX_OWNER	VARCHAR2(30)	Index owner
ERR_INDEX_NAME	VARCHAR2(30)	Name of index
ERR_TIMESTAMP	DATE	Time of error
ERR_TEXTKEY	VARCHAR2(18)	ROWID of errored document or name of errored operation (for example, ALTER INDEX)
ERR_TEXT	VARCHAR2 (4000)	Error text



F.12 CTX_INDEX_OBJECTS

This view displays the objects that are used for each class in the index. It can be queried by CTXSYS.

Column Name	Туре	Description	
IXO_INDEX_OWNER	VARCHAR2(30)	Index owner	
IXO_INDEX_NAME	VARCHAR2(30)	Index name	
IXO_CLASS	VARCHAR2(30)	Class name	
IXO_OBJECT	VARCHAR2(30)	Object name	

F.13 CTX_INDEX_PARTITIONS

This view displays all index partitions. It can be queried by CTXSYS.

Column Name	Туре	Description
IXP_ID	NUMBER(38)	Index partition ID
IXP_INDEX_OWNER	VARCHAR2(30)	Index owner
IXP_INDEX_NAME	VARCHAR2(30)	Index name
IXP_INDEX_PARTITION_NAME	VARCHAR2(30)	Index partition name
IXP_SYNC_TYPE	VARCHAR2(20)	Type of synching: MANUAL, AUTOMATIC, or ON COMMIT
IXP_TABLE_OWNER	VARCHAR2(30)	Table owner
IXP_TABLE_NAME	VARCHAR2(30)	Table name
IXP_TABLE_PARTITION_NAME	VARCHAR2(30)	Table partition name
IXP_DOCID_COUNT	NUMBER (38)	Number of documents associated with the partition
IXP_STATUS	VARCHAR2(12)	Partition status

F.14 CTX INDEX SETS

This view displays all index set names. It can be queried by any user.

Column Name	Туре	Description
IXS_OWNER	VARCHAR2(30)	Index set owner
IXS_NAME	VARCHAR2(30)	Index set name

F.15 CTX_INDEX_SET_INDEXES

This view displays all the sub-indexes in an index set. It can be queried by any user.



Column Name	Туре	Description
IXX_INDEX_SET_OWNER	VARCHAR2(30)	Index set owner
IXX_INDEX_SET_NAME	VARCHAR2(30)	Index set name
IXX_COLLIST	VARCHAR2(500)	Column list of the sub-index
IXX_STORAGE	VARCHAR2(500)	Storage clause of the sub-index

F.16 CTX_INDEX_SUB_LEXERS

This view shows the sub-lexers for each language for each index. It can be queried by CTXSYS.

Column Name	Туре	Description
ISL_INDEX_OWNER	VARCHAR2(30)	Index owner
ISL_INDEX_NAME	VARCHAR2(30)	Index name
ISL_LANGUAGE	VARCHAR2(30)	Language of sub-lexer
ISL_ALT_VALUE	VARCHAR2(30)	Alternate value of language
ISL_OBJECT	VARCHAR2(30)	Name of lexer object used for this language

F.17 CTX_INDEX_SUB_LEXER_VALUES

Shows the sub-lexer attributes and their values. Accessible by CTXSYS.

Column Name	Туре	Description
ISV_INDEX_OWNER	VARCHAR2(30)	Index owner
ISV_INDEX_NAME	VARCHAR2(30)	Index name
ISV_LANGUAGE	VARCHAR2(30)	Language of sub-lexer
ISV_OBJECT	VARCHAR2(30)	Name of lexer object used for this language
ISV_ATTRIBUTE	VARCHAR2(30)	Name of sub-lexer attribute
ISV_VALUE	VARCHAR2(500)	Value of attribute of sub-lexer

F.18 CTX_INDEX_VALUES

This view displays attribute values for each object used in indexes. This view is queryable by CTXSYS.

Column Name	Туре	Description	
IXV_INDEX_OWNER	VARCHAR2(30)	Index owner	
IXV_INDEX_NAME	VARCHAR2(30)	Index name	
IXV_CLASS	VARCHAR2(30)	Class name	
IXV_OBJECT	VARCHAR2(30)	Object name	
IXV_ATTRIBUTE	VARCHAR2(30)	Attribute name	



Column Name	Туре	Description
IXV_VALUE	VARCHAR2(500)	Attribute value

F.19 CTX_OBJECTS

This view displays all of the Text objects registered in the Text data dictionary. This view can be queried by any user.

Column Name	Туре	Description
OBJ_CLASS	VARCHAR2(30)	Object class (Datastore, Filter, Lexer, and so on)
OBJ_NAME	VARCHAR2(30)	Object name
OBJ_DESCRIPTION	VARCHAR2(80)	Object description

F.20 CTX_OBJECT_ATTRIBUTES

This view displays the attributes that can be assigned to preferences of each object. It can be queried by all users.

Column Name	Туре	Description
OAT_CLASS	VARCHAR2(30)	Object class (Data Store, Filter, Lexer, and so on)
OAT_OBJECT	VARCHAR2(30)	Object name
OAT_ATTRIBUTE	VARCHAR2(64)	Attribute name
OAT_DESCRIPTION	VARCHAR2(80)	Description of attribute
OAT_REQUIRED	VARCHAR2(1)	Required attribute, either Y or N
OAT_STATIC	VARCHAR2(1)	Not currently used
OAT_DATATYPE	VARCHAR2(64)	Attribute datatype. The value PROCEDURE indicates that the attribute of the object should be a stored procedure name.
OAT_DEFAULT	VARCHAR2(500)	Default value for attribute
OAT_MIN	NUMBER	Minimum value
OAT_MAX	NUMBER	Maximum value
OAT_MAX_LENGTH	NUMBER	Maximum length

F.21 CTX_OBJECT_ATTRIBUTE_LOV

This view displays the allowed values for certain object attributes provided by Oracle Text. It can be queried by all users.

Column Name	Туре	Description
OAL_CLASS	NUMBER(38)	Class of object
OAL_OBJECT	VARCHAR2(30)	Object name
OAL_ATTRIBUTE	VARCHAR2(32)	Attribute name



Column Name	Туре	Description
OAl_LABEL	VARCHAR2(30)	Attribute value label
OAL_VALUE	VARCHAR2 (64)	Attribute value
OAL_DESCRIPTION	VARCHAR2(80)	Attribute value description

F.22 CTX_ORDER_BY_COLUMNS

This view displays the ORDER BY columns registered in the Text data dictionary. It can be queried by any user.

Column Name	Туре	Description
OBC_INDEX_OWNER	VARCHAR2(30)	Index owner
OBC_INDEX_NAME	VARCHAR2(30)	Index name
OBC_TABLE_OWNER	VARCHAR2(30)	Table owner
OBC_TABLE_NAME	VARCHAR2(30)	Table name
OBC_COLUMN_NAME	VARCHAR2 (236)	Column name
OBC_COLUMN_POSITION	VARCHAR2(30)	Column position
OBC_COLUMN_TYPE	VARCHAR2(30)	Column type
OBC_SECTION_NAME	VARCHAR2(30)	Section name
OBC_SECTION_TYPE	VARCHAR2(30)	Section type
OBC_SECTION_ID	NUMBER	Section ID
OBC_SORT_ORDER	VARCHAR2(8)	Sort order

F.23 CTX_PARAMETERS

This view displays all system-defined parameters as defined by \mathtt{CTXSYS} . It can be queried by any user.

Column Name	Туре	Description
PAR_NAME	VARCHAR2(30)	Parameter name:
		auto_optimize
		auto_optimize_logfile
		max_index_memory
		ctx_doc_key_type
		default_index_memory
		default_datastore
		default_filter_binary
		default_filter_text
		default_filter_file
		default_section_html
		default_section_xml
		default_section_text
		default_lexer
		default_stoplist
		default_storage
		default_wordlist
		default_ctxcat_lexer
		default_ctxcat_index_set
		default_ctxcat_stoplist
		default_ctxcat_storage
		default_ctxcat_wordlist
		default_ctxrule_lexer
		default_ctxrule_stoplist
		default_ctxrule_storage
		default_ctxrule_wordlist
		log_directory
PAR_VALUE	VARCHAR2 (500)	Parameter value. For max_index_memory and default_index_memory, PAR_VALUE stores a string consisting of the memory amount. For the other parameter names, PAR_VALUE stores the names of the preferences used as defaults for index creation.

F.24 CTX_PREFERENCES

This view displays preferences created by Oracle Text users, as well as all the system-defined preferences included with Oracle Text. The view contains one row for each preference. It can be queried by all users.

Column Name	Туре	Description
PRE_OWNER	VARCHAR2(30)	Username of preference owner
PRE_NAME	VARCHAR2(30)	Preference name
PRE_CLASS	VARCHAR2(30)	Preference class
PRE_OBJECT	VARCHAR2(30)	Object used



F.25 CTX_PREFERENCE_VALUES

This view displays the values assigned to all the preferences in the Text data dictionary. The view contains one row for each value. It can be queried by all users.

Column Name	Туре	Description
PRV_OWNER	VARCHAR2(30)	Username of preference owner
PRV_PREFERENCE	VARCHAR2(30)	Preference name
PRV_ATTRIBUTE	VARCHAR2 (64)	Attribute name
PRV_VALUE	VARCHAR2(500)	Attribute value

F.26 CTX_SECTIONS

This view displays information about all the sections, including SDATA and MDATA sections, that have been created in the Text data dictionary. It can be queried by any user.

Column Name	Туре	Description
SEC_OWNER	VARCHAR2(30)	Owner of the section group
SEC_SECTION_GROUP	VARCHAR2(30)	Name of the section group
SEC_TYPE	VARCHAR2(30)	Type of section, either ZONE, FIELD, SPECIAL, ATTR, STOP
SEC_ID	NUMBER	Section ID
SEC_NAME	VARCHAR2(30)	Name of section
SEC_TAG	VARCHAR2(64)	Section tag
SEC_VISIBLE	VARCHAR2(1)	Y or N visible indicator for field sections only. Y indicator for READ ONLY MDATA sections.
SEC_DATATYPE	VARCHAR2(30)	Shows the datatype name (NUMBER, VARCHAR2, DATE or RAW) if the section is an SDATA section. Otherwise, it is NULL.

F.27 CTX_SECTION_GROUPS

This view displays information about all the section groups that have been created in the Text data dictionary. It can be queried by any user.

Column Name	Туре	Description	
SGP_OWNER	VARCHAR2(30)	Owner of section group	
SGP_NAME	VARCHAR2(30)	Name of section group	
SGP_TYPE	VARCHAR2(30)	Type of section group	

F.28 CTX_SQES

This view displays the definitions for all SQEs that have been created by users. It can be queried by all users.



Column Name	Туре	Description
SQE_OWNER	VARCHAR2(30)	Owner of SQE
SQE_NAME	VARCHAR2(30)	Name of SQE
SQE_QUERY	CLOB	Query Text

F.29 CTX_STOPLISTS

This view displays stoplists. Queryable by all users.

Column Name	Туре	Description
SPL_OWNER	VARCHAR2(30)	Owner of stoplist
SPL_NAME	VARCHAR2(30)	Name of stoplist
SPL_COUNT	NUMBER	Number of stopwords
SPL_TYPE	VARCHAR2(30)	Type of stoplist, MULTI or BASIC

F.30 CTX_STOPWORDS

This view displays the stopwords in each stoplist. Queryable by all users.

Column Name	Туре	Description
SPW_OWNER	VARCHAR2(30)	Stoplist owner
SPW_STOPLIST	VARCHAR2(30)	Stoplist name
SPW_TYPE	VARCHAR2(10)	Stop type, either STOP_WORD, STOP_CLASS, STOP_THEME
SPW_WORD	VARCHAR2(80)	Stopword
SPW_LANGUAGE	VARCHAR2(30)	Stopword language
SPW_PATTERN	VARCHAR2 (512)	Stop pattern

F.31 CTX_SUB_LEXERS

This view contains information on multi-lexers and the sub-lexer preferences they contain. It can be queried by any user.

Column Name	Туре	Description
SLX_OWNER	VARCHAR2(30)	Owner of the multi-lexer preference
SLX_NAME	VARCHAR2(30)	Name of the multi-lexer preference
SLX_LANGUAGE	VARCHAR2(30)	Language of the referenced lexer (full name, not abbreviation)
SLX_ALT_VALUE	VARCHAR2(30)	An alternate value for the language
SLX_SUB_OWNER	VARCHAR2(30)	Owner of the sub-lexer
SLX_SUB_NAME	VARCHAR2(30)	Name of the sub-lexer



F.32 CTX THESAURI

This view displays information about all the thesauri that have been created in the Text data dictionary. It can be queried by any user.

Column Name	Туре	Description
THS_OWNER	VARCHAR2(30)	Thesaurus owner
THS_NAME	VARCHAR2(30)	Thesaurus name

F.33 CTX_THES_PHRASES

This view displays phrase information for all thesauri in the Text data dictionary. It can be queried by any user.

Column Name	Туре	Description
THP_THESAURUS	VARCHAR2(30)	Thesaurus name
THP_PHRASE	VARCHAR2(256)	Thesaurus phrase
THP_QUALIFIER	VARCHAR2(256)	Thesaurus qualifier
THP_SCOPE_NOTE	VARCHAR2(2000)	Thesaurus scope notes

F.34 CTX_TRACE_VALUES

This view contains one row for each active trace, and shows the current value of each trace.

Column Name	Туре	Description
TRC_ID	BINARY_INTEGER	Trace ID
TRC_VALUE	NUMBER	Current trace value

Note:

The error "ORA-00955: name is already used by an existing object" can safely be ignored if this error is raised in the postinstall steps for patch releases. This may occur when this view is present in the database being patched.

F.35 CTX_USER_ALEXER_DICTS

This view displays all dictionaries created by the current user.

Column Name	Туре	Description
ALD_NAME	VARCHAR2(30)	Name of the dictionary
ALD_LANG	VARCHAR2(30)	Language of the dictionary



F.36 CTX_USER_AUTO_OPTIMIZE_INDEXES

This view displays the user indexes that are registered for auto optimization. It can be queried by all users.

Column Name	Туре	Description
AOI_INDEX_NAME	VARCHAR2(30)	Index name
AOI_PARTITION_NAME	VARCHAR2(30)	Partition name



In Oracle Database Release 21c, the procedures <code>ADD_AUTO_OPTIMIZE</code> and <code>REMOVE_AUTO_OPTIMIZE</code>, and the views <code>CTX_AUTO_OPTIMIZE_INDEXES</code>, <code>CTX_USER_AUTO_OPTIMIZE_INDEXES</code> and <code>CTX_AUTO_OPTIMIZE_STATUS</code> are deprecated.

F.37 CTX_USER_AUTOSYNC_JOBS

The CTX_USER_AUTOSYNC_JOBS view lists all background synchronization jobs that belong to the indexes in an Oracle Text user's schema.

Column Name	Туре	Description
ASJ_JOB_NAME	VARCHAR2 (128)	Name of the synchronization job
ASJ_INDEX_NAME	VARCHAR2 (128)	Name of the Oracle Text index
ASJ_INDEX_ID	NUMBER(38)	Index ID
ASJ_PARTITION_NAME	VARCHAR2(128)	Partition name
ASJ_PARTITION_ID	NUMBER	Partition ID
ASJ_START_DATE	TIMESTAMP(6) WITH TIME ZONE	Original start date of the synchronization job
ASJ_REPEAT_INTERVAL	VARCHAR2(4000)	Repeat interval
ASJ_STATE	VARCHAR2(15)	Current state of the job
ASJ_RUN_COUNT	NUMBER	Number of times the synchronization job has run

Note:

Synchronization jobs are now owned by the <code>CTXSYS</code> user. You can not use the <code>USER_SCHEDULER_JOBS</code> view to view the background synchronization jobs for indexes.

F.38 CTX USER AUTOSYNC STATUS

The CTX_USER_AUTOSYNC_STATUS view lists the status of each run of a background synchronization job for indexes in an Oracle Text user's schema.

Column Name	Туре	Description
ASJ_JOB_NAME	VARCHAR2 (1044)	Name of the synchronization job
ASJ_INDEX_NAME	VARCHAR2 (128)	Name of the Oracle Text index
ASJ_PARTITION_NAME	VARCHAR2 (128)	Partition name
ASJ_PARTITION_NAME	VARCHAR2 (128)	Oracle Text partition name
ASJ_TIMESTAMP	TIMESTAMP(6) WITH TIME ZONE	Date of the log entry
ASJ_STATUS	VARCHAR2(30)	Status of the synchronization job
ASJ_ERROR	VARCHAR2(4000)	Errors generated by the synchronization job



Synchronization jobs are now owned by the CTXSYS user. You can not use the USER_SCHEDULER_JOBS view to view the background synchronization jobs for indexes.

F.39 CTX_USER_BACKGROUND_EVENTS

The CTX_USER_BACKGROUND_EVENTS view displays historical information about the execution of background events for indexes with automatic maintenance. This view filters events for the current user based on the index owner.

A similar view CTX BACKGROUND EVENTS displays events for the SYS or CTXSYS user.

The objects are represented by both their numbers and names. However, because the data is historical, it is possible that those objects may no longer exist. Thus, the object names may be \mathtt{NULL} , however the object numbers are never \mathtt{NULL} .

Column Name	Туре	Description
BGE_OWNER#	NUMBER	Index owner number
BGE_OWNER_NAME	VARCHAR2 (128)	Index owner name
BGE_TABLE#	NUMBER	Base table object number
BGE_TABLE_NAME	VARCHAR2 (128)	Base table name
BGE_INDEX#	NUMBER	Index object number
BGE_INDEX_NAME	VARCHAR2 (128)	Index name
BGE_TABLE_PARTITION #	NUMBER	Base table partition object number
BGE_TABLE_PARTITION _NAME	VARCHAR2 (128)	Base table partition name



Column Name	Туре	Description
BGE_INDEX_PARTITION #	NUMBER	Index partition object number
BGE_INDEX_PARTITION _NAME	VARCHAR2 (128)	Index partition name
BGE_EVENT_TYPE	VARCHAR2 (30)	Event type: NONE SYNC-Mapping Timeout (Sync-MT) SYNC-Mapping (Sync-M) SYNC-Ranges (Sync-R) SYNC-Scheduler (Sync-S) SYNC-Postings Serial (Sync-PS) SYNC-Postings Concurrent (Sync-PC) SYNC-Writer (Sync-W) SYNC-Cleanup batches (Sync-C) SYNC-Inspect (Sync-I) MONITOR EVENT Stats (EStat) EVENT Stats Clean up (EClean) OPTIMIZE-Scheduler Timeout (Opti-ST) OPTIMIZE-Scheduler (Opti-S)
BGE_EVENT_ITERATION	NUMBER	Event retries happen with an increased delay up to a maximum of 10 hr. Depending on the event type, the retries could go through a single event loop or through a longer loop. For example, if Sync-W fails, then it needs to go through Sync-C, Sync-S, SYNC-Postings (Sync-P) before it can reach Sync-W again. Event iterations track the number of times a longer loop has been retried, and event waits track single event delays.
BGE_EVENT_WAIT	NUMBER	Time in minutes for which the event waited to retry before it was scheduled. The delays increase each time the event fails and max out at 10 hr. The delays are 3 sec, 10 sec, 30 sec, 1 min, 3 min, 10 min, 30 min, 1 hr, 3 hr, and 10 hr.
BGE WORKER NUMBER	NUMBER	Internal worker number
BGE BATCH NUMBER	NUMBER	Internal SGA batch number
BGE_STATUS	VARCHAR2(30)	 Event status: EMPTY: No status information RUNNING: Event is running DONE: Successful completion BUSY: Index or index partition is busy INTERRUPTED: Event is interrupted by DDL CLOSED: PDB is closed DROPPED: Index or index partition is dropped or altered FAILED: Event failed with an error FATAL: Worker process is terminated

Column Name	Туре	Description
BGE_START_TIME	TIMESTAMP(6) WITH TIMEZONE	Event processing start time
BGE_END_TIME	TIMESTAMP(6) WITH TIMEZONE	Event processing end time
BGE_DURATION	NUMBER	Event processing duration in seconds
BGE_ERROR_NUMBER	NUMBER	Top error number
BGE_ERROR_MESSAGE	VARCHAR2 (4000)	Complete stack of error messages
BGE_NUM_ROWS	NUMBER	 Sync-M: Number of processed DMLs Sync-R: number of selected NEW ranges Sync-S: Number of READY ranges Sync-P: Number of processed documents (document count) Sync-W: Number of documents written Sync-C: Number of ranges deleted Sync-I: Number of events enqueued Monitor: Number of objects processed
BGE_FIRST_DOCID	NUMBER	 Sync-M: NEW range first DocID Sync-R: First DocID of the first READY range Sync-S: First DocID of the first READY range Sync-P: First DocID of the selected READY range Sync-W: First DocID of the WRITE range Opti-S: First DocID of the aggregated MERGE range
BGE_CURRENT_DOCID	NUMBER	Sync-P: Current DocID of the selected READY range (last processed)
BGE_LAST_DOCID	NUMBER	 Sync-M: NEW range last DocID Sync-R: Last DocID of the last READY range Sync-S: Last DocID of the last READY range Sync-P: Last DocID of the selected READY range Sync-W: Last DocID of the WRITE range Opti-S: Last DocID of the aggregated MERGE range
BGE_BATCH_WAIT	NUMBER	Sync-P: Time waited for batch in minutes
BGE_BATCH_SIZE	NUMBER	Sync-P: Batch memory size in bytesSync-W: Batch memory size in bytes
BGE_DOCUMENT_SIZE	NUMBER	 Sync-R: Estimated per-document memory size Sync-R: Average per-document memory size
BGE_TOKEN_COUNT	NUMBER	Sync-P: Token countSync-W: Token countOpti-S: Token count
BGE_RANGE_COUNT	NUMBER	 Sync-M: Number of deleted DocIDs Sync-R: Number of generated READY ranges Sync-P: Generated READY range count



Column Name	Туре	Description	
BGE_RANGE_TYPE	NUMBER	 Sync-R: READY range type Sync-P: READY range type Ready range type: NA INITIAL RETRY 	

Related Topics

Oracle Text Application Developer's Guide

F.40 CTX_USER_EXTRACT_POLICIES

This view displays all of the entity extraction policies owned by the current user. All users can query this view.

Column Name	Туре	Description
EPL_NAME	VARCHAR2(30)	Entity extraction policy name

F.41 CTX_USER_EXTRACT_POLICY_VALUES

This view displays all of the values for the entity extraction policies owned by the current user. All users can query this view.

Column Name	Туре	Description
EPV_POLICY_NAME	VARCHAR2(30)	Entity extraction policy name
EPV_CLASS	VARCHAR2(30)	Object class
EPV_OBJECT	VARCHAR2(30)	Object name
EPV_ATTRIBUTE	VARCHAR2(30)	Object attribute name
EPV_VALUE	VARCHAR2 (500)	Object attribute value

F.42 CTX_USER_EXTRACT_RULES

This view displays the entity extraction rules for the policies owned by the current user. All users can query this view.

Column Name	Туре	Description
ERL_POLICY_NAME	VARCHAR2(30)	Entity extraction policy name
ERL_RULE_ID	INTEGER	Entity extraction rule ID
ERL_LANGUAGE	VARCHAR2(30)	Entity extraction rule language
ERL_RULE	VARCHAR2 (512)	Entity extraction rule contents
ERL_TYPE	VARCHAR2 (4000)	String mapping backreferences to entity types
ERL_STATUS	VARCHAR2(30)	Entity extraction rule status: compiled, not compiled, to be deleted



Column Name	Туре	Description
ERL_COMMENTS	VARCHAR2(4000)	Comments

F.43 CTX_USER_EXTRACT_STOP_ENTITIES

This view displays the stop entities owned by the current user. All users can query this view.

Column Name	Туре	Description
ESE_POLICY_NAME	VARCHAR2(30)	Entity extraction policy name
ESE_NAME	VARCHAR2 (512)	Stop entity name
ESE_TYPE	VARCHAR2(30)	Stop entity type
ESE_STATUS	VARCHAR2(30)	Entity extraction rule status: compiled, not compiled, to be deleted, subset
ESE_COMMENTS	VARCHAR2 (4000)	Comments

F.44 CTX_USER_EXTRACT_TYPE

The CTX_USER_EXTRACT_TYPE view displays the entity extraction types for the policies owned by the current user. All users can query this view.

Column Name	Туре	Description
ERT_POL_ID	INTEGER	Entity extraction policy ID
ERT_RULE_ID	INTEGER	Entity extraction rule ID
ERT_TYPE_ID	INTEGER	Entity extraction rule type ID
ERT_TYPE	VARCHAR2(512)	Entity extraction type

F.45 CTX_USER_ FILTER_BY_COLUMNS

This view displays all FILTER BY columns registered in the Text data dictionary for the current user. It can be queried by any user.

Column Name	Туре	Description
FBC_INDEX_NAME	VARCHAR2(30)	Index name
FBC_TABLE_OWNER	VARCHAR2(30)	Table owner name
FBC_TABLE_NAME	VARCHAR2(30)	Table name
FBC_COLUMN_NAME	VARCHAR2 (256)	Column name
FBC_COLUMN_TYPE	VARCHAR2(30)	Column type
FBC_SECTION_TYPE	VARCHAR2(30)	Section type
FBC_SECTION_NAME	VARCHAR2(30)	Section name
FBC_SECTION_ID	NUMBER	Section ID

F.46 CTX_USER_INDEXES

This view displays all indexes that are registered in the Text data dictionary for the current user. It can be queried by all users.

Column Name	Туре	Description
IDX_CHARSET_COLUMN	VARCHAR2 (256)	Name of the charset column of base table
IDX_DOCID_COUNT	NUMBER	Number of documents indexed
IDX_FORMAT_COLUMN	VARCHAR2 (256)	Name of the format column of base table
IDX_ID	NUMBER	Internal index ID
IDX_KEY_NAME	VARCHAR (256)	Primary key column(s)
IDX_LANGUAGE_COLUMN	VARCHAR2 (256)	Name of the language column of base table
IDX_NAME	VARCHAR2(30)	Name of index
IDX_STATUS	VARCHAR2(12)	Status, either INDEXED or INDEXING
IDX_SYNC_INTERVAL	VARCHAR2 (2000)	This is the interval string required by scheduler job. Only meaningful for AUTOMATIC sync. Always null for MANUAL and ON COMMIT sync.
IDX_SYNC_JOBNAME	VARCHAR2(50)	This is the scheduler job name for automatic sync. Only meaningful for AUTOMATIC sync and always null for other types of sync.
IDX_SYNC_MEMORY	VARCHAR2 (100)	The sync memory size. Only meaningful for ON COMMIT and AUTOMATIC types of sync. For MANUAL sync, this is always null.
IDX_SYNC_PARA_DEGREE	NUMBER	Degree of parallelism for sync. Only meaningful for the AUTOMATIC type of sync; always null for MANUAL and ON COMMIT syncs.
IDX_SYNC_TYPE	VARCHAR2 (20)	Type of synching: AUTOMATIC, MANUAL or ON COMMIT
IDX_TABLE	VARCHAR2(30)	Table name
IDX_TABLE_OWNER	VARCHAR2(30)	Owner of table
IDX_TEXT_NAME	VARCHAR2(30)	Text column name
IDX_TYPE	VARCHAR2(30)	Type of index: CONTEXT, CTXCAT, or CTXRULE

F.47 CTX_USER_INDEX_ERRORS

This view displays the indexing errors for the current user and is queryable by all users.

Column Name	Туре	Description
ERR_INDEX_NAME	VARCHAR2(30)	Name of index
ERR_TIMESTAMP	DATE	Time of error
ERR_TEXTKEY	VARCHAR2(18)	ROWID of errored document or name of errored operation (for example, ALTER INDEX)
ERR_TEXT	VARCHAR2(4000)	Error text

F.48 CTX_USER_INDEX_OBJECTS

This view displays the preferences that are attached to the indexes defined for the current user. It can be queried by all users.

Column Name	Туре	Description	
IXO_INDEX_NAME	VARCHAR2(30)	Name of index	
IXO_CLASS	VARCHAR2(30)	Object name	
IXO_OBJECT	VARCHAR2(80)	Object description	

F.49 CTX_USER_INDEX_PARTITIONS

This view displays all index partitions for the current user. It is queryable by all users.

Column Name	Туре	Description
IXP_DOCID_COUNT	NUMBER(38)	Number of documents associated with the index partition
IXP_ID	NUMBER (38)	Index partition ID
IXP_INDEX_NAME	VARCHAR2(30)	Index name
IXP_INDEX_PARTITION_NAME	VARCHAR2(30)	Index partition name
IDX_SYNC_INTERVAL	VARCHAR2 (2000)	This is the interval string required by scheduler job. Only meaningful for AUTOMATIC sync. Always null for MANUAL and ON COMMIT sync.
IDX_SYNC_JOBNAME	VARCHAR2 (50)	This is the scheduler job name for automatic sync. It is only meaningful for AUTOMATIC sync and always null for other types of sync.
IDX_SYNC_MEMORY	VARCHAR2 (100)	The sync memory size. Only meaningful for ON COMMIT and AUTOMATIC types of sync. For MANUAL sync, this is always null.
IDX_SYNC_PARA_DEGREE	NUMBER	Degree of parallelism for sync. Only meaningful for the AUTOMATIC type of sync; always null for MANUAL and ON COMMIT syncs.
IDX_SYNC_TYPE	VARCHAR2(20)	Type of synching: AUTOMATIC, MANUAL or ON COMMIT
IXP_STATUS	VARCHAR2(12)	Partition status
IXP_TABLE_OWNER	VARCHAR2(30)	Table owner



Column Name	Туре	Description
IXP_TABLE_NAME	VARCHAR2(30)	Table name
IXP_TABLE_PARTITION_NAME	VARCHAR2(30)	Table partition name

F.50 CTX_USER_INDEX_SETS

This view displays all index set names that belong to the current user. It is queryable by all users.

Column Name	Туре	Description
IXS_NAME	VARCHAR2(30)	Index set name

F.51 CTX_USER_INDEX_SET_INDEXES

This view displays all the indexes in an index set that belong to the current user. It is queryable by all users.

Column Name	Туре	Description
IXX_INDEX_SET_NAME	VARCHAR2(30)	Index set name
IXX_COLLIST	VARCHAR2(500)	Column list of the index
IXX_STORAGE	VARCHAR2(500)	Storage clause of the index

F.52 CTX_USER_INDEX_SUB_LEXERS

This view shows the sub-lexers for each language for each index for the querying user. This view can be queried by all users.

Column Name	Туре	Description
ISL_INDEX_NAME	VARCHAR2(30)	Index name
ISL_LANGUAGE	VARCHAR2(30)	Language of sub-lexer
ISL_ALT_VALUE	VARCHAR2(30)	Alternate value of language
ISL_OBJECT	VARCHAR2(30)	Name of lexer object used for this language

F.53 CTX_USER_INDEX_SUB_LEXER_VALS

Shows the sub-lexer attributes and their values for the querying user. This view can be queried by all users.

Column Name	Туре	Description
ISV_INDEX_NAME	VARCHAR2(30)	Index name
ISV_LANGUAGE	VARCHAR2(30)	Language of sub-lexer
ISV_OBJECT	VARCHAR2(30)	Name of lexer object used for this language

Column Name	Туре	Description	
ISV_ATTRIBUTE	VARCHAR2(30)	Name of sub-lexer attribute	
ISV_VALUE	VARCHAR2 (500)	Value of sub-lexer attribute	

F.54 CTX_USER_INDEX_VALUES

This view displays attribute values for each object used in indexes for the current user. This view is queryable by all users.

Column Name	Туре	Description
IXV_INDEX_NAME	VARCHAR2(30)	Index name
IXV_CLASS	VARCHAR2(30)	Class name
IXV_OBJECT	VARCHAR2(30)	Object name
IXV_ATTRIBUTE	VARCHAR2(30)	Attribute name
IXV_VALUE	VARCHAR2 (500)	Attribute value

F.55 CTX_USER_ORDER_BY_COLUMNS

This view displays all ORDER BY columns registered in the Text data dictionary for the current user. It can be queried by any user.

Column Name	Туре	Description
OBC_INDEX_NAME	VARCHAR2(30)	Index name
OBC_TABLE_OWNER	VARCHAR2(30)	Table owner
OBC_TABLE_NAME	VARCHAR2(30)	Table name
OBC_COLUMN_NAME	VARCHAR2 (236)	Column name
OBC_COLUMN_POSITION	VARCHAR2(30)	Column position
OBC_COLUMN_TYPE	VARCHAR2(30)	Column type
OBC_SECTION_NAME	VARCHAR2(30)	Section name
OBC_SECTION_TYPE	VARCHAR2(30)	Section type
OBC_SECTION_ID	NUMBER	Section ID
OBC_SORT_ORDER	VARCHAR2(8)	Sort order

F.56 CTX_USER_PREFERENCES

This view displays all preferences defined by the current user. It can be queried by all users.

Column Name	Туре	Description
PRE_NAME	VARCHAR2(30)	Preference name
PRE_CLASS	VARCHAR2(30)	Preference class
PRE_OBJECT	VARCHAR2(30)	Object used



F.57 CTX_USER_PREFERENCE_VALUES

This view displays all the values for preferences defined by the current user. It can be queried by all users.

Column Name	Туре	Description
PRV_PREFERENCE	VARCHAR2(30)	Preference name
PRV_ATTRIBUTE	VARCHAR2(64)	Attribute name
PRV_VALUE	VARCHAR2(500)	Attribute value

F.58 CTX_USER_SECTIONS

This view displays information about the sections that have been created in the Text data dictionary for the current user. It can be queried by all users.

Column Name	Туре	Description
SEC_DATATYPE	VARCHAR2(30)	Shows the datatype name (NUMBER, VARCHAR2, DATE or RAW) if the section is an SDATA section. Otherwise, it is NULL.
SECSECTION_GROUP	VARCHAR2(30)	Name of the section group
SEC_TYPE	VARCHAR2(30)	Type of section, either ZONE, FIELD, SPECIAL, STOP, or ATTR
SEC_ID	NUMBER	Section ID
SEC_NAME	VARCHAR2(30)	Name of section
SEC_TAG	VARCHAR2(64)	Section tag
SEC_VISIBLE	VARCHAR2(1)	Y or N visible indicator for field sections

F.59 CTX_USER_SECTION_GROUPS

This view displays information about the section groups that have been created in the Text data dictionary for the current user. It can be queried by all users.

Column Name	Туре	Description
SGP_NAME	VARCHAR2(30)	Name of section group
SGP_TYPE	VARCHAR2(30)	Type of section group

F.60 CTX_USER_SESSION_SQES

This view displays the definitions of all session-duration SQEs that have been created by the current user.

Column Name	Туре	Description
SQE_OWNER	VARCHAR2(30)	Name of owner of SQE



Column Name	Туре	Description
SQE_NAME	VARCHAR2(30)	Name of SQE (shared namespace between persistent and session-duration)
SQE_QUERY	CLOB	Query text (max size of 32k)

F.61 CTX_USER_SQES

This view displays the definitions of all persistent duration SQEs that have been created by the current user. In other words, it does not display session duration SQEs.

Column Name	Туре	Description	
SQE_OWNER	VARCHAR2(30)	Owner of SQE	
SQE_NAME	VARCHAR2(30)	Name of SQE	
SQE_QUERY	CLOB	Query text	

F.62 CTX_USER_STOPLISTS

This view displays stoplists for current user. It is queryable by all users.

Column Name	Туре	Description
SPL_NAME	VARCHAR2(30)	Name of stoplist
SPL_COUNT	NUMBER	Number of stopwords
SPL_TYPE	VARCHAR2(30)	Type of stoplist, MULTI or BASIC

F.63 CTX_USER_STOPWORDS

This view displays stopwords in each stoplist for current user. Queryable by all users.

Column Name	Туре	Description
SPW_STOPLIST	VARCHAR2(30)	Stoplist name
SPW_TYPE	VARCHAR2(10)	Stop type, either STOP_WORD, STOP_CLASS, STOP_THEME
SPW_WORD	VARCHAR2(80)	Stopword
SPW_LANGUAGE	VARCHAR2(30)	Stopword language
SPW_PATTERN	VARCHAR2(512)	Stop pattern

F.64 CTX_USER_SUB_LEXERS

For the current user, this view contains information on multi-lexers and the sub-lexer preferences they contain. It can be queried by any user.



Column Name	Туре	Description
SLX_NAME	VARCHAR2(30)	Name of the multi-lexer preference
SLX_LANGUAGE	VARCHAR2(30)	Language of the referenced lexer (full name, not abbreviation)
SLX_ALT_VALUE	VARCHAR2(30)	An alternate value for the language
SLX_SUB_OWNER	VARCHAR2(30)	Owner of the sub-lexer
SLX_SUB_NAME	VARCHAR2(30)	Name of the sub-lexer

F.65 CTX_USER_THESAURI

This view displays the information about all of the thesauri that have been created in the system by the current user. It can be viewed by all users.

Column Name	Туре	Description
THS_NAME	VARCHAR2(30)	Thesaurus name

F.66 CTX_USER_THES_PHRASES

This view displays the phrase information of all thesauri owned by the current user. It can be queried by all users.

Column Name	Туре	Description
THP_THESAURUS	VARCHAR2(30)	Thesaurus name
THP_PHRASE	VARCHAR2(256)	Thesaurus phrase
THP_QUALIFIER	VARCHAR2(256)	Phrase qualifier
THP_SCOPE_NOTE	VARCHAR2(2000)	Scope note of the phrase

F.67 CTX_VERSION

This view displays the CTXSYS data dictionary and code version number information.

Column Name	Туре	Description
VER_DICT	CHAR(9)	The CTXSYS data dictionary version number
VER_CODE	VARCHAR2(9)	The version number of the code linked in to the Oracle Database shadow process
		This column fetches the version number for linked-in code. Thus, use this column to detect and verify patch releases.



F.68 ALL_DBMS_SEARCH_INDEXES

The ALL_DBMS_SEARCH_INDEXES view displays information about all existing DBMS_SEARCH indexes, corresponding to each index owner.

Column Name	Туре	Description
IDX_OWNER	VARCHAR2 (128)	Schema owner name associated with the ubiquitous search index
IDX_NAME	VARCHAR2(128)	Ubiquitous search index name

F.69 ALL_DBMS_SEARCH_INDEX_SOURCES

The ALL_DBMS_SEARCH_INDEX_SOURCES view displays information about all existing data sources added to various DBMS_SEARCH indexes, corresponding to each index owner.

Column Name	Туре	Description
IDX_OWNER	VARCHAR2 (128)	Schema owner name associated with the ubiquitous search index
IDX_NAME	VARCHAR2 (128)	Ubiquitous search index name
SRC_OWNER	VARCHAR2 (128)	Data source owner name
SRC_NAME	VARCHAR2 (128)	Data source name
SRC_TYPE	VARCHAR2(1)	Data source type, such as: T: Table source V: View source J: JSON duality view source
SRC_ID	NUMBER	Source object ID

F.70 USER_DBMS_SEARCH_INDEXES

The <code>USER_DBMS_SEARCH_INDEXES</code> view displays information about the <code>DBMS_SEARCH</code> indexes that are created in a user's schema.

Column Name	Туре	Description
IDX_NAME	VARCHAR2(128)	Ubiquitous search index name

F.71 USER_DBMS_SEARCH_INDEX_SOURCES

The <code>USER_DBMS_SEARCH_INDEX_SOURCES</code> view displays information about the data sources that are added to the <code>DBMS_SEARCH</code> indexes, created in a user's schema.

Column Name	Туре	Description
IDX_NAME	VARCHAR2(128)	Ubiquitous search index name
SRC_OWNER	VARCHAR2(128)	Data source owner name
SRC_NAME	VARCHAR2(128)	Data source name



Column Name	Туре	Description
SRC_TYPE	VARCHAR2(1)	Data source type, such as:
		T: Table source
		V: View source
		 J: JSON duality view source
SRC_ID	NUMBER	Source object ID



G

Stopword Transformations in Oracle Text

This appendix describes the stopword rewrites or transformations for each operator. In all tables, the *Stopword Expression* column describes the query expression or component of a query expression, while the right-hand column describes the way Oracle Text rewrites the query.

This appendix contains the following topics:

- Understanding Stopword Transformations
- About Stopwords in Phrase Queries
- Word Transformations
- AND Transformations
- OR Transformations
- ACCUMulate Transformations
- MINUS Transformations
- MNOT Transformations
- NOT Transformations
- EQUIValence Transformations
- NEAR Transformations
- Weight Transformations
- · Threshold Transformations
- WITHIN Transformations

G.1 Understanding Stopword Transformations

When you use a stopword or stopword-only phrase as an operand for a query operator, Oracle Text rewrites the expression to eliminate the stopword or stopword-only phrase and then executes the query.

The token *stopword* stands for a single stopword or a stopword-only phrase.

The token *non_stopword* stands for either a single non-stopword, a phrase of all non-stopwords, or a phrase of non-stopwords and stopwords.

The token *no_lex* stands for a single character or a string of characters that is neither a stopword nor a word that is indexed. For example, the + character by itself is an example of a *no_lex* token.

When the *Stopword Expression* column completely describes the query expression, a rewritten expression of *no_token* means that no hits are returned when you enter such a query.

When the *Stopword Expression* column describes a component of a query expression with more than one operator, a rewritten expression of *no_token* means that a *no_token* value is passed to the next step of the rewrite.

Transformations that contain a *no_token* as an operand in the *Stopword Expression* column describe intermediate transformations in which the *no_token* is a result of a previous transformation. These intermediate transformations apply when the original query expression has at least one stopword and more than one operator.

For example, consider the following compound query expression:

'(this NOT dog) AND cat'

Assuming that *this* is the only stopword in this expression, Oracle Text applies the following transformations in the following order:

stopword NOT non-stopword => no_token

no_token AND non_stopword => non_stopword

The resulting expression is:

'cat'

G.2 About Stopwords in Phrase Queries

If used in a phrase query, a stopword will match any single word, whether that word is a stopword or not. For example, if "in" and "to" are stopwords, but "throughout" is not, then the query "hiking in California" will match any of these phrases:

- hiking in California
- hiking to California
- hiking throughout California

G.3 Word Transformations

Stopword Expression	Rewritten Expression
stopword	no_token
no_lex	no_token

The first transformation means that a stopword or stopword-only phrase by itself in a query expression results in no hits.

The second transformation says that a term that is not lexed, such as the + character, results in no hits.

G.4 AND Transformations

Stopword Expression	Rewritten Expression
non_stopword AND stopword	non_stopword
non_stopword AND no_token	non_stopword
stopword AND non_stopword	non_stopword
no_token AND non_stopword	non_stopword
stopword AND stopword	no_token
no_token AND stopword	no_token



Stopword Expression	Rewritten Expression
stopword AND no_token	no_token
no_token AND no_token	no_token

G.5 OR Transformations

Stopword Expression	Rewritten Expression
non_stopword OR stopword	non_stopword
non_stopword OR no_token	non_stopword
stopword OR non_stopword	non_stopword
no_token OR non_stopword	non_stopword
stopword OR stopword	no_token
no_token OR stopword	no_token
stopword OR no_token	no_token
no_token OR no_token	no_token

G.6 ACCUMulate Transformations

Stopword Expression	Rewritten Expression
non_stopword ACCUM stopword	non_stopword
non_stopword ACCUM no_token	non_stopword
stopword ACCUM non_stopword	non_stopword
no_token ACCUM non_stopword	non_stopword
stopword ACCUM stopword	no_token
no_token ACCUM stopword	no_token
stopword ACCUM no_token	no_token
no_token ACCUM no_token	no_token

G.7 MINUS Transformations

Stopword Expression	Rewritten Expression
non_stopword MINUS stopword	non_stopword
non_stopword MINUS no_token	non_stopword
stopword MINUS non_stopword	no_token
no_token MINUS non_stopword	no_token
stopword MINUS stopword	no_token
no_token MINUS stopword	no_token
stopword MINUS no_token	no_token
no_token MINUS no_token	no_token



G.8 MNOT Transformations

Stopword Expression	Rewritten Expression
non_stopword MNOT stopword	non_stopword
non_stopword MNOT no_token	non_stopword
stopword MNOT non_stopword	no_token
no_token MNOT non_stopword	no_token
stopword MNOT stopword	no_token
no_token MNOT stopword	no_token
stopword MNOT no_token	no_token
no_token MNOT no_token	no_token

G.9 NOT Transformations

Stopword Expression	Rewritten Expression
non_stopword NOT stopword	non_stopword
non_stopword NOT no_token	non_stopword
stopword NOT non_stopword	no_token
no_token NOT non_stopword	no_token
stopword NOT stopword	no_token
no_token NOT stopword	no_token
stopword NOT no_token	no_token
no_token NOT no_token	no_token

G.10 EQUIValence Transformations

Stopword Expression	Rewritten Expression
non_stopword EQUIV stopword	non_stopword
non_stopword EQUIV no_token	non_stopword
stopword EQUIV non_stopword	non_stopword
no_token EQUIV non_stopword	non_stopword
stopword EQUIV stopword	no_token
no_token EQUIV stopword	no_token
stopword EQUIV no_token	no_token
no_token EQUIV no_token	no_token





When you use query explain plan, not all of the equivalence transformations are represented in the ${\tt EXPLAIN}$ table.

G.11 NEAR Transformations

Stopword Expression	Rewritten Expression
non_stopword NEAR stopword	non_stopword
non_stopword NEAR no_token	non_stopword
stopword NEAR non_stopword	non_stopword
no_token NEAR non_stopword	non_stopword
stopword NEAR stopword	no_token
no_token NEAR stopword	no_token
stopword NEAR no_token	no_token
no_token NEAR no_token	no_token

G.12 Weight Transformations

Stopword Expression	Rewritten Expression
stopword * n	no_token
no_token * n	no_token

G.13 Threshold Transformations

Stopword Expression	Rewritten Expression
stopword > n	no_token
no_token > n	no_token

G.14 WITHIN Transformations

Stopword Expression	Rewritten Expression
stopword WITHIN section	no_token
no_token WITHIN section	no_token

